Pattern Discovery in Data Streams under the Time Warping Distance

Machiko Toyoda · Yasushi Sakurai · Yoshiharu Ishikawa

Abstract Subsequence matching is a basic problem in the field of data stream mining. In recent years, there has been significant research effort spent on efficiently finding subsequences similar to a query sequence. Another challenging issue in relation to subsequence matching is how we identify common local patterns when both sequences are evolving. This problem arises in trend detection, clustering, and outlier detection. Dynamic time warping (DTW) is often used for subsequence matching and is a powerful similarity measure. However, the straightforward method using DTW incurs a high computation cost for this problem. In this paper, we propose a one-pass algorithm, CrossMatch, that achieves the above goal. CrossMatch addresses two important challenges: (1) how can we identify common local patterns efficiently without any omission? (2) how can we find common local patterns in data stream processing? To tackle these challenges, CrossMatch incorporates three ideas: (1) a scoring function, which computes the DTW distance indirectly to reduce the computation cost, (2) a position matrix, which stores starting positions to keep track of common local patterns in a streaming fashion, and (3) a streaming algorithm, which identifies common local patterns efficiently and outputs them on the fly. We provide a theoretical analysis and prove that our algorithm does not sacrifice accuracy. Our experimental evaluation and case studies show that CrossMatch can incrementally discover common local patterns in data streams within constant time (per update) and space.

Machiko Toyoda
NTT Communication Science Laboratories, Kyoto, Japan
E-mail: toyoda.machiko@lab.ntt.co.jp

Yasushi Sakurai
NTT Communication Science Laboratories, Kyoto, Japan
E-mail: yasushi.sakurai@acm.org

Yoshiharu Ishikawa
Information Technology Center, Nagoya University, Aichi, Japan
E-mail: ishikawa@itc.nagoya-u.ac.jp

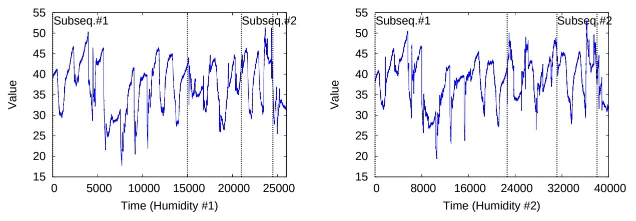
1 Introduction

Data streams are becoming increasingly important in several domains including financial data analysis [68], sensor network monitoring [74], moving object trajectories [15, 38], web click-stream analysis [42,48,59], and network traffic analysis [37]. Many applications require time-series data streams to be continuously monitored in real time, and the processing and mining of data streams are attracting increasing interest. In addition to providing SQL-like support for data stream management systems (DSMS), it is crucial to detect hidden patterns that may exist in data streams, and subsequence matching is one of the key techniques for achieving this goal.

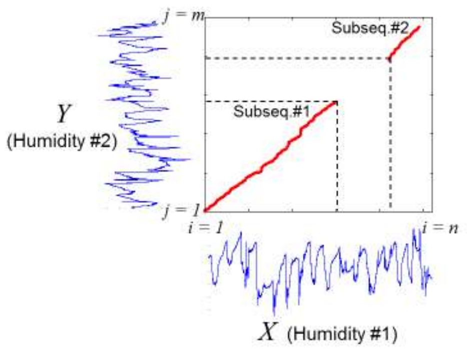
Much of the previous work on subsequence matching over data streams has focused on finding subsequences similar to a query sequence [16,58,72]. In this setting, one is a fixed sequence and the other is an evolving sequence. This approach works well if we have already determined the patterns we want to find. However, we consider co-evolving sequences and focus on the problem of identifying common local patterns between them. That is, our goal is to automatically detect all common local patterns over data streams without a query sequence. The problem we want to solve is as follows.

Given two data streams, determine common local patterns and their periodicities taking account of time scaling.

This problem is defined in detail in Section 3, in Fig. 1 we present a visual intuition. The two datasets in Fig. 1 (a) show humidity readings from two different sensors. The sensors send the readings approximately every minute. These sequences are roughly similar, but a part of the sequences is different. Intuitively, the problem is to identify the partial similarity of the sequences through data stream processing. In this example, we discover two common patterns marked with vertical lines: Subseq. #1 and #2. The matches between



(a) Data sequences



(b) Detected common local patterns

Fig. 1 Illustration of problem. Two humidity sequences have two common local patterns. Our proposed method identifies their positions and similarities in a streaming fashion.

the two patterns are shown in Fig. 1 (b). The lines represent the matches between the elements of X and Y and correspond to a diagonal if the two subsequences match perfectly. We identify their similarities and matching points, i.e., the starting and end positions of each sequence, in a streaming fashion, and report each match as early as possible.

This problem motivates us to develop the following important techniques: (1) trend detection, which is the ability to detect the most frequently occurring patterns in data streams, (2) clustering, which is the ability to find sequences that look similar and to group them, and (3) outlier detection, which is the ability to discover anomalous patterns by comparing common patterns. These exciting techniques could also provide interpretations of clusters and anomalies by annotating them in an online fashion.

In addition to the above techniques, we also consider the following interesting applications.

- Web analysis: Web access patterns are very dynamic because of both the dynamics of web site content and structure, and the changes in the users' interests. A continuous monitoring of web access will reveal interesting usage patterns or profiles and provide users with more suitable, customized services in real time. Webmasters may cluster users into groups based on their common characteristics for user behavioral analysis. Web site designers can use typical browsing patterns to personalize the user's experience on the website. These groups and patterns essentially correspond to groups of common local patterns.
- Motion capture: The recognition of human motion has been attracting intense interest in relation to computer animation, sports, and medical care. Motion data sequences are sampled many times per second and are data streams of high dimensionality. Humans never repeat exactly the same action patterns, and the actions tend to differ in terms of their duration. This appears as variability in the speed of human motion. For example, an actor may walk quickly or slowly. Such variability can manifest itself as time scaling, namely a stretching or shrinking of time-series data. Our approach aids trend detection, which can be used to identify particular movement styles for game creators, and outlier detection, which can be used by coaches to analyze athletes' performance by identifying time-varying common motions (i.e., common local patterns).
- Sensor network: In sensor networks, sensors send their readings frequently. Each sensor produces a stream of data, and those streams need to be monitored and combined to detect interesting changes in the environment. It is likely that users are interested in one or more sensors within a particular spatial region. These interests are expressed as trends and similar patterns, i.e., common local patterns.

What similarity measures are suitable for detecting common local patterns? There are a large number of similarity measures for time-series analysis[22]. Unlike the traditional setting, data streams arrive continuously. Subsequence matching should focus on asynchronous data because streams frequently have different sampling rates. The mechanism should be robust against noise and provide scaling of the time axis. We use dynamic time warping (DTW) [9,54] to solve this problem. DTW is a robust and widely used measure in several domains [32,34,45]. It is also suitable for subsequence matching since it provides time scaling (such as the stretching or shrinking of a portion of a sequence along the time axis) [4,35,61,71,75].

What are the significant challenges in terms of detecting common local patterns over data streams? Typically, DTW is applied to limited situations in an offline manner. To identify common local patterns with DTW, we have to divide data streams into all possible subsequences and compute the similarities between them because we have no advance knowledge about the patterns we are seeking. Since data streams arrive online at high bit rates and are potentially unbounded in size, the computation time and memory space increase greatly. Ideally, we need a solution that can return correct results without any omissions, even at high speeds.

Recently, the work in [65,66] addressed the problem of finding common local patterns in data streams. Problem definition and its solution using DTW are introduced in [66], but the work does not provide any theoretical guarantees with respect to answer accuracy and it output overlapping results. In [65] we modified the problem definition and devised two ideas, a scoring function (Section 4.2.1) and a position matrix (Section 4.2.2). In this paper, while we share the same goals, we present a new streaming algorithm (Section 4.2.3), which incorporates these ideas and at the same time provides strict guarantees for our results (Section 4.3). By introducing a global constraint for DTW, which is suitable for stream settings, our algorithm improves the time and space requirements. Moreover, we propose enhanced solutions for different environments (Sections 5 and 6) and make our algorithm more robust.

Our contributions in this paper are as follows.

- We present CrossMatch, which can efficiently detect common local patterns in data streams. CrossMatch is a one-pass algorithm, which is strictly based on DTW and guarantees correct results.
- In our theoretical analysis, we prove that CrossMatch does not sacrifice accuracy and detects the optimal subsequences. Moreover, we discuss the complexity in terms of computation time and memory space and show that CrossMatch significantly reduces the required amounts of these resources and achieves constant time (per update) and space.
- For more effectiveness, we propose a sampling approach that introduces an approximation for CrossMatch. Our solution works properly for sampled sequences and achieves a significant reduction in resources.
- As regards the accuracy and complexity for detecting common local patterns, we empirically show its usefulness on several real and synthetic datasets.
- We address a more challenging problem of finding common local patterns in multiple data streams, and show that CrossMatch can be effectively applied to this problem.

The remainder of this paper is organized as follows. Section 2 discusses related work and Section 3 provides the problem definition. In Section 4 we describe the ideas behind CrossMatch and its algorithm, and in Section 5 we introduce a sampling approach for CrossMatch. Section 6 presents an enhanced algorithm for multiple streams. Section 7 reviews our experimental results, which clearly demonstrate the effectiveness of CrossMatch. Section 8 provides our conclusion.

2 Related Work

Related work falls broadly into three categories: time-series similarity search, stream management and stream mining. We review each category.

Time-series analysis and similarity search. Time-series analysis has been studied for many years. Most of the proposed methods focus on similarity queries with a query sequence. There are several distance measures for similarity queries on time-series data, e.g., euclidean distance [25], dynamic time warping (DTW) [9,54], distance based on the longest common subsequence (LCSS) [67], edit distance with real penalty (EDP) [14], and edit distance on real sequence (EDR) [15]. These distance measures are selected depending on the difference of the matching strategy in application domains.

To efficiently perform the similarity search efficiently, data sequences are transformed to lower dimensional points with a dimensionality reduction technique. Agrawal et al. [2] and Faloutsos et al. [25] have utilized discrete Fourier transformation (DFT), and have inserted each point into an R-tree [8]. Other reduction techniques include discrete wavelet transform (DWT) [53], singular value decomposition (SVD) [33], piecewise aggregate approximation (PAA) [70], and adaptive piecewise constant approximation (APCA) [36]. Cao et al. [10] have proposed a data reduction technique for spatio-temporal data.

Sequence matching has attracted a lot of research interest, and very successful methods have been developed for time-series data [4,35,61]. MDMWP [31] is a fast ranked subsequence matching solution. Ranked subsequence matching finds the top-k similar subsequences to a query sequence from data sequences. It introduces two tight lower bounds and prunes unnecessary subsequence access requests at the index level. EBSM [5] is a method for approximate subsequence matching under DTW. The key idea is to convert subsequence matching to vector matching. For the conversion, EBSM uses precomputed alignments between database sequences and query sequences. Rakthanmanon et al. [55] have focused on one trillion length time-series and several different many tens of billions time-series data and have proposed a method for searching exactly under DTW. By introducing the four optimizations based on the early stop of the computation and lower bounds, they have shown that their method is much faster than the recent search method for DTW. The above methods focus mainly on stored sequences.

As regards subsequence matching based on DTW in data streams, Zhou et al. [72] presented an efficient batch filtering method. They observe a special property of data streams, which is that successive subsequences in a stream often overlap to some extent, and improve the performance by utilizing such overlapping information as filters for lower and upper bounds. Sakurai et al. [58] presented SPRING, which efficiently monitors multiple numerical streams. They introduce two new ideas; star-padding and subsequence time warping matrix. These methods can accurately detect similar subsequences in a constant time without fixing the window size. On the other hand, Chen et al. [16] proposed an original distance function that supports shifting and scaling in both the time and amplitude dimensions and used it as a similarity measure for the efficient and continuous detection of patterns in a streaming time sequence. The above methods are powerful for dealing with problems where fixed length query sequences are given. However, they scale poorly and so are ineffective with respect to our target problem.

Regarding the detection of common local patterns over data streams, the most relevant work is [66], which proposed an algorithm for finding similar subsequences. From an algorithmic perspective, a highly relevant work is [65], which presents an algorithm based on DTW. However, the algorithm in [66] does not guarantee the correctness of results and outputs subsequences with redundant information. Both algorithms are linear with regard to time and space. This paper not only overcomes the issues of accuracy and complexity, but is also more efficient in detecting common local patterns.

In the field of bioinformatics, search techniques for biological sequences have been studied and the Smith-Waterman algorithm is used to find local similarities [63]. The studies in this area focus on symbol sequences. Whereas, our problem focuses on numerical sequences. Our method differs in that it computes the DTW distance precisely and guarantees the detection of subsequences with the minimum distance.

Continuous queries and data stream management. Broadly related work includes data stream management systems (DSMSs). Their common goal is to provide a general-purpose infrastructure for the efficient management of data streams. Sample systems include *Aurora* [1], *Stream* [44], *Telegraph* [12], *Gigascope* [19], and *OSCAR* [13]. Algorithmic work includes query processing [41], scheduling [6,11], and load shedding [20,64]. As regards continuous queries, Arasu et al. [3] studied the memory requirements of continuous queries over *relational* data streams. SOLE [43] is a scalable algorithm for continuous spatio-temporal queries in data streams. To address multiple streams and queries, it provides a framework with caching of uncertainty regions and a shared operator on a shared buffer.

Approximation and adaptivity are also key features for DSMSs, such as sampling [7], sketches [17,23,27], statistics [21,28], and wavelets [30]. The main goal of these methods is to estimate a global aggregate (e.g., sum, count, average) over a fixed window on the recent data.

The emphasis in the above works is to support traditional SQL queries on streams. None of them try to find patterns.

Stream mining. Many other previous studies have attempted pattern discovery in a streaming scenario. Mueen et al. [46] presented the first online motif discovery algorithm to accurately monitor and maintain motifs, which represent repeated subsequences in time-series, in real time. AWSOM [50] is one of the first streaming algorithms for forecasting, and it is used to discover arbitrary periodicities in a time sequence. Zhu et al. [73] focused on monitoring multiple streams in real time and proposed StatStream, which computes pairwise correlations among all streams. The SPIRIT method [51] is used to address the problem of capturing correlations and finding hidden variables corresponding to trends in collections of co-evolving data streams. BRAID [60] detects lag correlations between data streams by using geometric probing and smoothing to approximate the exact correlation. Papadimitriou et al. [52] proposed an algorithm for discovering optimal local patterns, which concisely describe the main trends in data streams. DynaMMo [40] summarizes and compresses multiple sequences and finds latent variables among them.

On the other hand, there are effective methods that address massive time-series streams as applications for data center management. Reeves et al. [56] addressed the problem of the space-efficient archiving of time-series streams and the fast processing of several statistical and data mining queries regarding that archived data. They focused on the problem that traditional database systems have addressed space-efficient archiving and query processing separately, and proposed Cypress, which preprocesses and decomposes each data stream into a small number of substreams, and answers common queries directly from a set of them rather than reconstructing the original stream. Mueen et al. [47] considered the problem of computing all-pair correlations in a warehouse containing a large number of time-series. A high I/O and CPU overhead make the fast computation of correlations a challenging issue. They proposed a caching algorithm to optimize overall I/O cost and two approximation algorithms to reduce CPU costs.

These techniques focus on trend detection, correlation, motif discovery, and prediction, and so are not solutions for our goal, which is to find common local patterns based on DTW.

In our experiment on CrossMatch, we used scatter plots to show its outputs, which were the optimal subsequence pairs. Recurrence plot [24] and dot plot [69] have been proposed for visual sequence analysis and mining of time-series data; they focus on the visualization of the similar parts of sequences on a scatter plot. Our objective is to identify which of the subsequences of X and Y are similar by applying the DTW approach in an online manner, and so differs from their objective and approach.

3 Problem definition

In this section, we introduce dynamic time warping (DTW) [9,54], and then define the problem that forms our objective. The main symbols used in this paper are shown in Table 1.

Table 1 Definitions of main symbols

Symbols	Definitions
X	Data sequence/stream of length n
Y	Data sequence/stream of length m
x_i	i-th element of X
y_j	j-th element of Y
$X[i_s : i_e]$ $Y[j_s : j_e]$	Subsequences of X and Y, including elements in positions i_s, j_s through i_e, j_e
ε	Distance threshold for finding qualifying subsequences
l_{min}	Threshold of subsequence length
l_x	Length of $X[i_s : i_e]$
l_y	Length of $Y[j_s : j_e]$
$L(l_x, l_y)$	Function for length between $X[i_s : i_e]$ and $Y[j_s : j_e]$
w	Width of warping scope
$d(i, j)$	Distance of (i, j) in time warping matrix
$v(i, j)$	Score of (i, j) in score matrix
$s(i, j)$	Starting position of (i, j) in position matrix
$\mathcal{X}$	Sampled data sequence of X
$\mathcal{Y}$	Sampled data sequence of Y
x_i	i-th element of $\mathcal{X}$
y_j	j-th element of $\mathcal{Y}$
f_{Nq}	Nyquist frequency
T_x	Fixed sampling period of X
T_y	Fixed sampling period of Y

3.1 Preliminaries

DTW is a transformation that allows sequences to be stretched along the time axis to minimize the distance between them (see Fig.2). The DTW distance of two sequences is the sum of the tick-to-tick distances after the two sequences have been optimally warped to match each other. To align two sequences, we construct a '*time warping matrix*'. The warping path is a set of grid cells in the time warping matrix, which represents the alignment between the sequences. Consider two sequences, $X = (x_1, x_2, ..., x_n)$ of length n and $Y = (y_1, y_2, ..., y_m)$ of length m. Their DTW distance $D(X, Y)$ is defined as

$$D(X, Y) = d(n, m)$$

$$d(i, j) = ||x_i - y_j|| + \min \begin{cases} d(i, j-1) \\ d(i-1, j) \\ d(i-1, j-1) \end{cases} \qquad (1)$$

$$d(0, 0) = 0, \quad d(i, 0) = d(0, j) = \infty$$
$$(i = 1, ..., n;\ j = 1, ..., m).$$

Note that $||x_i - y_j|| = (x_i - y_j)^2$ is the distance between two numerical values in cell (i, j) of the time warping matrix. Note that other choices (say, absolute difference $||x_i - y_j|| = |x_i - y_j|$) can also be used; our algorithm is completely independent of the choice made. To avoid degenerated matching, where a relatively small section of one sequence maps onto a relatively large section of another, the warping path is limited by global constraints. The warping scope w is the area that the warping path is allowed to visit in the time warping matrix. The Sakoe-Chiba band [57] is a

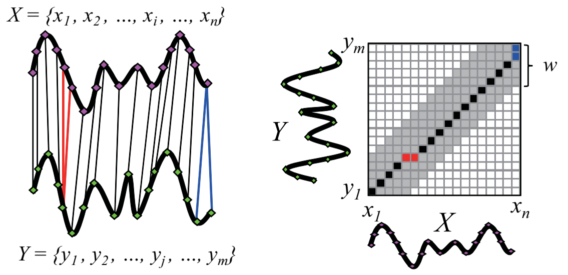

$X = \{x_1, x_2, ..., x_i, ..., x_n\}$
$Y = \{y_1, y_2, ..., y_j, ..., y_m\}$
y_m
w
Y
y_1
x_1
X
x_n

Fig. 2 Illustration of DTW. The left figure indicates the alignment of measurements. The right figure indicates the optimal warping path in a warping scope.

well-known global constraint that restricts the warping path to the range of $|i - j| \leq w$.

DTW requires $O(nm)$ time since the time warping matrix consists of nm cells. Note that the space complexity is $O(m)$ (or $O(n)$) since the algorithm needs only two columns (i.e., the current and previous columns) of the time warping matrix to compute the DTW distance. By using the warping scope, the time complexity is reduced to $O(nw + mw)$. The space complexity is $O(w)$ because we need only $2w$ cells.

3.2 Cross-similarity

Data stream X is a discrete, semi-infinite sequence of numbers $x_1, x_2, \ldots, x_n, \ldots$, where x_n is the most recent value. Note that n increases with every new time-tick. Let $X[i_s : i_e]$ be the subsequence of X that starts from time-tick i_s and ends at i_e, and let $Y[j_s : j_e]$ be the subsequence of Y that starts from time-tick j_s and ends at j_e. The lengths of $X[i_s : i_e]$ and $Y[j_s : j_e]$ are $l_x = i_e - i_s + 1$ and $l_y = j_e - j_s + 1$, respectively. Our goal is to find the common local patterns of sequences by data stream processing based on DTW. That is, we want to detect subsequence pairs that satisfy

$$D(X[i_s : i_e], Y[j_s : j_e]) \leq \varepsilon L(l_x, l_y), \qquad (2)$$

where $D(X[i_s : i_e], Y[j_s : j_e])$ is the DTW distance between $X[i_s : i_e]$ and $Y[j_s : j_e]$, ε is a distance threshold, and L is a function that sets the length of the subsequence. In this paper, the algorithm uses $L(l_x, l_y) = (l_x + l_y)/2$, which is the average length of the two subsequences, but the user can employ any other choice (e.g., $L(l_x, l_y) = max(l_x, l_y)$ or $L(l_x, l_y) = min(l_x, l_y)$). The DTW distance increases as the subsequence length increases since it is the sum of the distances between elements. Therefore, the distance threshold should be proportional to the subsequence length. Accordingly, we set it at $\varepsilon L(l_x, l_y)$.

Equation (2) allows us to detect subsequence pairs without regard to the subsequence length. In practice, however, we might detect shorter and meaningless matching pairs due to the influence of noise. We introduce the concept of subsequence match length to enable us to discard such meaningless pairs and to detect the optimal pairs that satisfy 'real' user requirements. We formally define the '*cross-similarity*' between X and Y, which indicates common local patterns.

Definition 1 (Cross-similarity [65]) Given two sequences X and Y, a distance threshold ε, and a threshold of subsequence length l_{min}, $X[i_s : i_e]$ and $Y[j_s : j_e]$ have the property of cross-similarity if this sequence pair satisfies the following condition.

$$D(X[i_s : i_e], Y[j_s : j_e]) \leq \varepsilon(L(l_x, l_y) - l_{min}). \tag{3}$$

The minimum length l_{min} of subsequence matches should be given by the users. The subsequences that satisfy this equation are guaranteed to have lengths exceeding l_{min}. We also agree that the user should select the length function L as well as l_{min} to obtain desirable results.

We should also mention the following point: whenever a subsequence pair matches, there will be several other matches that strongly overlap the 'local minimum' best match. Specifically, an overlap is simply the relation that two subsequence pairs have a common alignment, which is defined as follows:

Definition 2 (Overlap) Given two warping paths for subsequence pairs of X and Y, their overlap is defined as the condition where the paths share at least one element.

Overlaps provide the user with redundant information, and would slow down the algorithm since all useless 'solutions' are tracked and reported. Our solution is to detect the local best subsequences from the set of overlapping subsequences. Thus, our goal is to find the best match of cross-similarity.

Problem 1 Given two sequences X and Y, thresholds ε, and l_{min}, report all subsequence pairs, $X[i_s : i_e]$ and $Y[j_s : j_e]$, that satisfy the following conditions.

1. $X[i_s : i_e]$ and $Y[j_s : j_e]$ have the property of cross-similarity.
2. $D(X[i_s : i_e], Y[j_s : j_e]) - \varepsilon(L(l_x, l_y) - l_{min})$ is the minimum value among the set of overlapping subsequence pairs that satisfies the first condition.

Hereafter we use 'qualifying' subsequence pairs to refer to pairs that satisfy the first condition, and we use 'optimal' subsequence pairs to refer to pairs that satisfy both conditions.

Typically, new elements in data streams, i.e., those that have occurred recently, are usually more significant than those in the distant past [18]. To limit the cell in the matrix and focus on recent elements, we utilize a concept of global constraint for DTW, namely the Sacoe-Chiba band [57]. More specifically, for each sequence X and Y, we compute the cells from the recent element (e.g., x_n or y_m) to an element of the warping scope w ago. If $m = n$, the warping scope is exactly equal to the Sakoe-Chiba band.

4 Proposed method

In this section, we describe a straightforward solution to find the best match of cross-similarity in data streams and also present our one-pass algorithm, CrossMatch.

4.1 Naive solution

The most straightforward solution to this problem is to consider all possible subsequences of $X[i_s : i_e]$ $(1 \leq i_s < i_e \leq n)$, and all possible subsequences of $Y[j_s : j_e]$ $(1 \leq j_s < j_e \leq m)$ in the warping scope and apply the standard DTW dynamic programming algorithm. We call this method *Naive*.

Let $d_{i,j}(p, q)$ be the distance of cell (p, q) in the time warping matrix that starts from i on the x-axis and j on the y-axis, and let w be the width of the warping scope. The distance of the subsequence matching between X and Y can be obtained as follows.

$$\begin{aligned}
&D(X[i_s : i_e], Y[j_s : j_e]) = d_{i_s,j_s}(l_x, l_y) \\
&d_{i,j}(p,q) = ||x_{i+p-1} - y_{j+q-1}|| + \min \begin{cases} d_{i,j}(p, q-1) \\ d_{i,j}(p-1, q) \\ d_{i,j}(p-1, q-1) \end{cases} \\
&d_{i,j}(0,0) = 0, \quad d_{i,j}(p,0) = d_{i,j}(0,q) = \infty \\
&(i = 1, ..., n;\ p = 1, ..., n-i+1; \\
&\ j = 1, ..., m;\ q = 1, ..., m-j+1; \\
&\ n - w \leq i + p \leq n;\ m - w \leq j + q \leq m;)
\end{aligned} \tag{4}$$

The naive solution creates a new matrix at every new time-tick and updates the distance arrays of incoming x_i at time-tick i and that of incoming y_j at time-tick j in each existing time warping matrix. It then determines the subsequence pair for which $D(X[i_s : i_e], Y[j_s : j_e]) - \varepsilon(L(l_x, l_y) - l_{min})$ is the minimum value among the set of overlapping subsequence pairs.

Fig. 3 shows an example of a naive solution to the problem of subsequence matching. Let w be the width of the warping scope (the gray cell in the figure). The naive solution updates $O(w)$ distance values per time-tick on each matrix when an element of the sequence arrives. The naive solution requires $O(nw^2 + mw^2)$ time (per update) and space because it has to handle a total of $O(nw + mw)$ matrices to compute the DTW distance. In practice, it is not feasible to compute the distance in a streaming setting.

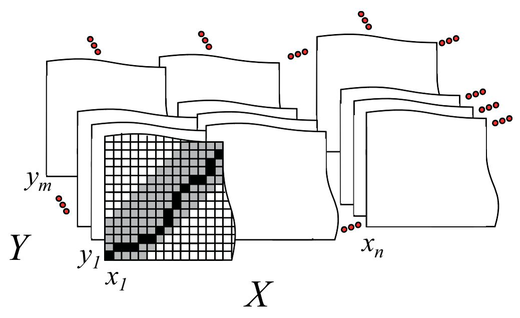


Fig. 3 Illustration of subsequence matching using the naive solution. The naive solution maintains matrices starting from every time-tick.

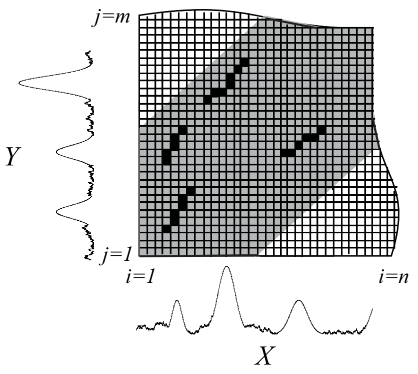


Fig. 4 Illustration of CrossMatch. The black cells indicate the warping paths of the optimal subsequence pairs and the gray cells indicate the warping scope.

4.2 CrossMatch

As mentioned in the previous section, the naive solution creates too many matrices because it computes the distance values between all possible subsequences. The distance threshold is proportional to the subsequence length (cf. Definition 1). The naive solution attempts to find the subsequence pairs semipermanently in each matrix. If we prune dissimilar subsequence pairs and reduce the number of matrices, the distance computations become much more efficient. Our method is motivated by this idea.

Our method, CrossMatch, computes the similarity score that corresponds to the DTW distance and identifies dissimilar subsequences. We find 'good' matches in a single matrix efficiently by pruning the subsequences (see Fig. 4). Our method, which realizes these concepts, consists of three ideas: a new scoring function, a position matrix, and a streaming algorithm that uses them.

4.2.1 Scoring function

To identify the dissimilar subsequences early, we propose computing the DTW distance indirectly by using a *scoring function*. The scoring function has the following two characteristics: (a) it provides a non-negative cumulative score, and (b) its operation is reversible with respect to the DTW distance.

The scoring function is essentially based on the dynamic programming approach. Whereas the DTW computes the minimum cumulative distance, our function computes the *maximum cumulative score* corresponding to the DTW distance with a *score matrix*. The score is determined by accumulating the difference between the threshold and the distance between the elements in the score matrix. Thus, we can recognize a dissimilar subsequence pair since the score has a negative value if the subsequence pair does not satisfy the first condition of Problem 1.

The scoring function selects the cell with the maximum cumulative score from the neighboring cells, and if the score is negative, the function initializes the score to zero and then restarts the computation from the cell. This operation allows us to discard unqualifying, non-optimal subsequence pairs.

Definition 3 (Score matrix [65]) Given two sequences, $X = (x_1, \ldots, x_i, \ldots, x_n)$ and $Y = (y_1, \ldots, y_j, \ldots, y_m)$, and the width of the warping scope w, score $V(X[i_s : i_e], Y[j_s : j_e])$ of $X[i_s : i_e]$ and $Y[j_s : j_e]$ is defined as:

$$V(X[i_s : i_e], Y[j_s : j_e]) = v(i_e, j_e)$$
$$v(i,j) = \max \begin{cases} 0 \\ \varepsilon b_v - ||x_i - y_j|| + v(i, j-1) \\ \varepsilon b_h - ||x_i - y_j|| + v(i-1, j) \\ \varepsilon b_d - ||x_i - y_j|| + v(i-1, j-1) \end{cases} \quad (5)$$
$$v(0,0) = v(i,0) = v(0,j) = 0$$
$$(i = 1, ..., n;\ j = 1, ..., m;$$
$$n - w \le i \le n;\ m - w \le j \le m;).$$

The scoring function operation is reversible with respect to the DTW distance. That is, the score of the qualifying subsequence pair with a positive value is easily transformed into the DTW distance. Symbols b_v, b_h, and b_d in Equation (5) indicate a weight function for each direction, which makes transformation between the score and the DTW distance possible. These values are determined as a function of the subsequence length. For example, for $L(l_x, l_y) = (l_x + l_y)/2$, the current L value increases by $1/2$ if the score of a vertical or horizontal cell is chosen, and it increases by 1 if the score of a diagonal cell is chosen. Thus, we obtain $b_v = b_h = 1/2$ and $b_d = 1$, respectively, for these directions [1]. The scoring

[1] For $L(l_x, l_y) = max(l_x, l_y)$, each weight is set as follows.
$b_d = b_h = 1$ and $b_v = 0$ if $l_x > l_y$.
$b_d = b_v = 1$ and $b_h = 0$ if $l_x < l_y$.
$b_d = 1$ and $b_v = b_h = 0$ if $l_x = l_y$.
Formally, each weight is defined as follows.
$b_v = L(l_x, l_y) - L(l_x, l_y - 1)$.
$b_h = L(l_x, l_y) - L(L_x - 1, l_y)$.
$b_d = L(l_x, l_y) - L(l_x - 1, l_y - 1)$.

6	13			21	0	0	0
5	9		16	22	25	20	0
4	2	11	0	0	27	**49**	0
3	4	13	0	0	36	42	0
2	9	0	11	26	24	0	
1	11	0	13	19	1		
j	y_j \ x_i	5	12	10	6	3	18
	i	1	2	3	4	5	6

(a) Score matrix

6	13			(1,3)	(4,6)	(5,6)	(6,6)
5	9		(1,3)	(1,3)	(2,1)	(2,1)	(6,5)
4	2	(1,3)	(2,4)	(3,4)	(2,1)	**(2,1)**	(6,4)
3	4	(1,3)	(2,3)	(3,3)	(2,1)	(2,1)	(6,3)
2	9	(1,2)	(2,1)	(2,1)	(2,1)	(5,2)	
1	11	(1,1)	(2,1)	(2,1)	(2,1)		
j	y_j \ x_i	5	12	10	6	3	18
	i	1	2	3	4	5	6

(b) Position matrix

Fig. 5 Example of cross-similarity detection. The light cells signify cross-similarity, and the dark cell in each matrix shows the best match.

function is designed so that the sum of the weights on the warping path (i.e., $b_v,\ b_h,$ and b_d) is equal to subsequence length L. Therefore, it guarantees reversibility between the DTW distance and the score, and finds the qualifying subsequence pairs without any omissions. The DTW distance of a subsequence pair is computed from the score and the subsequence length as follows:

$$D(X[i_s\!:\!i_e],Y[j_s\!:\!j_e])=\varepsilon L(l_x,l_y)-V(X[i_s\!:\!i_e],Y[j_s\!:\!j_e]) \quad s.t.\ V(X[i_s\!:\!i_e]),Y[j_s\!:\!j_e]) > 0. \tag{6}$$

Equation (6) holds for the time warping and the score matrices, which have the same starting position (i_s, j_s). The details are provided in Section 4.3.

Example 1 Assume that we have two sequences of $X = (5, 12, 6, 10, 3, 18)$, $Y = (11, 9, 4, 2, 9, 13)$, and $\varepsilon = 14$, $l_{min} = 2$, and $w = 3$. Fig. 5 (a) shows the score matrix. The dark cell, which has the highest score, shows the optimal subsequence pair and indicates that the score is $\varepsilon b_d - ||x_5 - y_4|| + v(4, 3) = 49$ and the end position is $(i_e, j_e) = (5, 4)$. The light cells show qualifying subsequence pairs. The cells that contain zero identify dissimilar subsequence pairs.

4.2.2 Position matrix

The scoring function tells us (a) where the subsequence match ends and (b) what the resulting score is. However, we lose the information about the starting position of the subsequence. This is the motivation behind our second idea, a *position matrix*: we store the starting position to keep track of the qualifying subsequence pair in a streaming fashion.

Definition 4 (Position matrix [65]) The position matrix stores the starting position of each subsequence pair. The starting position $s(i, j)$ corresponding to score $v(i, j)$ is computed as follows:

$$s(i,j)=\begin{cases} s(i,j-1) & (v(i,j\text{–}1)\neq 0 \wedge v(i,j) \\ & \quad =\varepsilon b_v-||x_i-y_j||+v(i,j-1)) \\ s(i-1,j) & (v(i\text{–}1,j)\neq 0 \wedge v(i,j) \\ & \quad =\varepsilon b_h-||x_i-y_j||+v(i-1,j)) \\ s(i-1,j-1) & (v(i\text{–}1,j\text{–}1)\neq 0 \wedge v(i,j) \\ & \quad =\varepsilon b_d-||x_i-y_j||+v(i-1,j-1)) \\ (i,j) & (otherwise). \end{cases} \tag{7}$$

The starting position is described as a coordinate value; $s(i_e, j_e)$ indicates the starting position (i_s, j_s) of the subsequence pair $X[i_s : i_e]$ and $Y[j_s : j_e]$. We update the starting position in the position matrix as well as the score in the score matrix. We can identify the optimal subsequence that gives the maximum score during stream processing since exactly the same warping path is maintained in the score and position matrices. Moreover, the starting position of the shared cell is maintained through the subsequent alignments because we repeat the operation, which maintains the starting position of the selected previous cell. Thus, we know the overlapping subsequence pairs from the fact that the starting positions match.

Example 2 Fig. 5 (b) shows the position matrix corresponding to the score matrix in Fig. 5 (a). In cell $(5, 4)$, the starting position $(2, 1)$ is maintained because the scoring function selects the score of cell $(4, 3)$ in the score matrix. By combining both matrices, we can identify the position of the optimal subsequence pair $X[2 : 5]$ and $Y[1 : 4]$. On the other hand, there are many overlapping subsequence pairs that have the same starting position $(2, 1)$. Of these, we select the subsequence pair with the highest score as the optimal pair because we can determine the overlapping subsequence pairs from the position matrix.

Next, we show how subsequence pairs are pruned. The pruned subsequence pairs fall into one of the following two categories: (1) subsequence pairs that are absolutely not reflected in two matrices (i.e., the score and the position matrices), and (2) subsequence pairs that are pruned during the computation process. In any case, our method is designed so that we can evaluate the cross-similarity between sequences from the score value, and guarantees that the pruned subsequence pairs are not optimal by using the fact that the overlapping subsequence pairs in cell (i, j) have the same warping paths in the subsequent alignments (we will provide detailed proofs in Section 4.3).

Example 3 An example of case (1) corresponds to the subsequence pairs starting at $(3, 2)$ in Fig. 5. In cell $(3, 2)$, our method has to select one pair from neighboring cells since all neighboring cells include positive scores, and it prunes the subsequence pairs that have the starting position $(3, 2)$. An example of case (2) corresponds to the subsequence pairs starting at $(1, 3)$. In cell $(4, 5)$, our method chooses the subsequence pair starting at $(2, 1)$ because the pair has the maximum value. That is, the subsequence pair that has the starting position $(1, 3)$ is pruned although the score indicates a positive value.

4.2.3 Streaming algorithm

We now have all the pieces needed to answer the question: how do we find the optimal subsequence pairs? Every time x_n is received at time-tick n, our algorithm, CrossMatch, incrementally updates the score $C'_v = v(n, j)$ and starting position $C'_s = s(n, j)$ and retains the end position $C'_e = (n, j)$. We use candidate array $\mathcal{S}$ to find the optimal subsequence pair and store the best pair C (i.e., score C_v, starting position C_s, and end position C_e) in a set of overlapping subsequence pairs. CrossMatch reports the optimal subsequence pair after confirming that it cannot be replaced by the upcoming subsequence pairs (i.e., there are no overlapping subsequence pairs). The upcoming candidate subsequence pairs do not overlap the captured optimal subsequence pair if the starting positions in the position matrix satisfy the following condition.

$$(\forall_i, s(i, m) \neq C_s) \ \wedge \ (\forall_j, s(n, j) \neq C_s).$$

CrossMatch reports the similarity of the subsequence pair as the DTW distance. The DTW distance is obtained from the score and the subsequence length, as shown in Equation (6).

The above procedure provides the foundation of our efficient detection of similar pairs. Algorithm 1 shows the details. We keep only two columns (i.e., the current and previous columns) for each X and Y in the two matrices. In this algorithm, we focus on computing the scores and the starting positions when we receive x_n at time-tick n. Note that the scores and the starting positions of incoming y_m at time-tick m are also computed similarly by this algorithm.

CrossMatch requires three parameters, l_{min}, w, and ε. The subsequence length l_{min} and the parameter ε are set based on the pattern the user wants to search. It is desirable to set the values according to the applications. The warping scope w determines the computation range in each matrix. At the same time, it asks the user how far back into the past the algorithm needs to go. If the user wants to search for subsequence pairs during the present time-tick and a time-tick in the relatively distant past, it is better to set a large w. In our experiments, we simply use reasonable values for every dataset, and we show that this way of setting parameters is sufficient for CrossMatch to verify the detection of the optimal subsequence pairs.

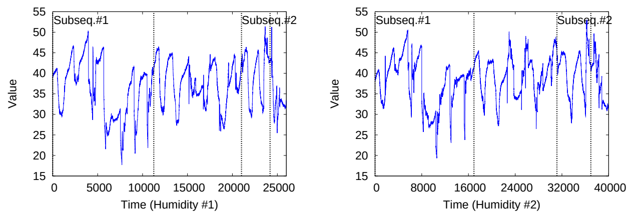
Algorithm 1 CrossMatch

```
Input: new value $x_n$ at time-tick $n$
Output: optimal subsequence pairs and DTW distances
 1: // Detect optimal subsequence pairs.
 2: for $j := m - w$ to $m$ do
 3:     $C'_v := v(n, j)$; // Score value derived by Equation (5)
 4:     $C'_s := s(n, j)$; // Starting position derived by Equation (7)
 5:     $C'_e := (n, j)$; // End position
 6:     if $C'_v \geq \varepsilon l_{min}$ then
 7:         // Add the subsequence pair as a new candidate.
 8:         if $C'_s \notin \mathcal{S}$ then
 9:             add $C'_v$, $C'_s$, and $C'_e$ to $\mathcal{S}$;
10:         else
11:             for each candidate $C \in \mathcal{S}$ do
12:                 // Overwrite the maximum score.
13:                 if $C'_s = C_s \wedge C'_v \geq C_v$ then
14:                     $C_v := C'_v$;
15:                     $C_e := C'_e$;
16:                 end if
17:             end for
18:         end if
19:     end if
20: end for
21: // Report the optimal subsequence pairs.
22: for each candidate $C \in \mathcal{S}$ do
23:     if $(\forall_i, s(i, m) \neq C_s) \wedge (\forall_j, s(n, j) \neq C_s)$ then
24:         $d_{min} = \varepsilon L(l_x, l_y) - C_v$;
25:         Report $d_{min}$, $C_s$ and $C_e$;
26:         Remove $C$ from $\mathcal{S}$;
27:     end if
28: end for
```

Example 4 Again assume two sequences of $X = (5, 12, 6, 10, 3, 18)$, $Y = (11, 9, 4, 2, 9, 13)$, and $\varepsilon = 14$, $l_{min} = 2$, and $w = 3$ in Fig 5. To simplify the example of our algorithm with no loss of generality, we assume that x_i and y_j arrive in alternately. At each time-tick, the algorithm updates the scores and the starting positions. At $i = 4$, we update the cells from $(4, 1)$ to $(4, 3)$ and identify a candidate subsequence, $X[2 : 4]$ and $Y[1 : 3]$, starting at $(2, 1)$, whose score $v(4, 3) = 36$ is greater than εl_{min}. At $j = 4$, we update the cells from $(1, 4)$ to $(4, 4)$. Although no subsequences satisfying the condition are detected, we do not report the subsequence of $X[2 : 4]$ and $Y[1 : 3]$ since it is possible that this pair could be replaced by upcoming subsequences. We then capture the optimal subsequence pair of $X[2 : 5]$ and $Y[1 : 4]$ at $i = 5$. We finally report the subsequence as the optimal subsequence at $j = 6$ since we can confirm that none of the upcoming subsequences can be optimal. Fig. 6 shows time warping matrix starting at $(2, 1)$ in the naive solution, which includes the optimal subsequence pair in Fig. 5. In the score and the position matrices, the subsequence pairs that have the starting position $(2, 1)$ correspond to the pairs on the time warping matrix in Fig. 6.

6	13			112	80	131	69
5	9		183	103	31	43	88
4	2		174	102	22	7	263
3	4		74	38	6	7	203
2	9		10	2	11	47	
1	11		1	2	27		
j	y_j / x_i	5	12	10	6	3	18
	i	1	2	3	4	5	6

Fig. 6 Time warping matrix starting at (2,1) in the naive solution.



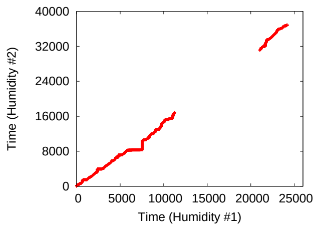


Fig. 7 Discovery of foremost subsequence pairs using *Humidity*. The subsequence pairs in the middle of alignments are detected, unlike the optimal subsequence pairs in Fig. 1.

From Equation (6), we have $\varepsilon L(4,4) - V(X[2:5], Y[1:4]) = 14 \cdot 4 - 49 = 7 = D(X[2:5], Y[1:4])$.

In this paper, we focus on finding only the optimal subsequence pairs. We provide one alternative with regard to the output of similar subsequence pairs. In a stream setting, it is desirable to report similar subsequence pairs as soon as possible. To report the similar subsequence pairs without delay, we firstly report the foremost subsequence pair, i.e., the first pair satisfying the threshold among the set of overlapping pairs, and thereafter update the pair with the optimal subsequence pair. For example, Fig. 7 shows the foremost subsequence pairs for the *Humidity* dataset in Fig. 1. We show the detailed comparison of their positions in Table 2. Note that (i_s, j_s) is the starting position and (i_e, j_e) is the end position. Unlike Fig. 1, it is obvious to shorten the reporting time. Thus, we can provide a solution that is more suitable for a streaming scenario.

4.3 Theoretical Analysis

We introduce a brief theoretical analysis that confirms the accuracy and complexity of CrossMatch.

4.3.1 Accuracy

Lemma 1 *Given two sequences X and Y, Problem 1 is equivalent to the following conditions.*

1. $V(X[i_s:i_e], Y[j_s:j_e]) \geq \varepsilon l_{min}$
2. $V(X[i_s:i_e], Y[j_s:j_e]) - \varepsilon l_{min}$ *is the maximum value in each group of subsequence pairs that the warping path crosses.*

Proof See appendix A. □

Lemma 2 *CrossMatch guarantees the output of the optimal subsequence pairs.*

Proof See appendix B. □

4.3.2 Complexity

Let X and Y be evolving sequences of lengths n and m, respectively.

Lemma 3 *The naive solution requires $O(nw^2 + mw^2)$ time (per update) and space to discover cross-similarity.*

Proof See appendix C. □

Lemma 4 *CrossMatch requires $O(w)$ (i.e., constant) time (per update) and space to discover cross-similarity.*

Proof See appendix D. □

5 Sampling approach

As mentioned above, CrossMatch detects cross-similarity in constant time and space. The next question is what we can do in the highly likely case that the users need more efficient solutions given that, in practice, they require high accuracy, not a theoretical guarantee. This is our motivation for introducing an approximation for CrossMatch.

What approximate techniques are suitable for CrossMatch? An efficient idea involves the data reduction of a sequence. Optimal alignments of DTW correspond to matching the elements in time. To find optimal subsequence pairs by approximation, we choose to keep the sequence, which is transformed by data reduction operated in the time domain. As an extended version of CrossMatch, we propose compressing the matrices using a *sampling approach*. As we show later, this decision significantly improves both space cost and response time, with negligible effect on the mining results.

As the first step, we consider the following theorem.

Theorem 1 (Sampling theorem) *If a continuous function contains no frequencies higher than f_{high}, it is completely determined by its value at a series of points less than $1/2f_{high}$ apart.*

Table 2 Comparison with positions of optimal and foremost subsequence pairs.

	Subseq. #1				Subseq. #2			
	i_s	i_e	j_s	j_e	i_s	i_e	j_s	j_e
Optimal subsequence pairs	1	15000	6	22629	21008	24505	31146	38013
Foremost subsequence pairs	1	11250	6	16938	21008	24188	31146	36949

Proof See [39]. □

In the theorem, the minimum sampling frequency, $f_{Nq} = 2f_{high}$, is called the Nyquist frequency. We utilize this theorem for sampling the sequences. That is, we use coarse sequences yielded by sampling based on the theorem, and detect the cross-similarity. Since the original sequence is sampled once for each f_{Nq} value, i.e., $T = 1/f_{Nq}$, we greatly reduce in the size of the matrix.

5.1 Scoring function

How do we compute the score between sampled sequences? Intuitively, the key idea is that when we select one of the neighboring cells for score computation, we interpolate the distance values that were dropped by sampling. In the score computation between sampled sequences, there are $T-1$ hidden cells that represent the missing values between the current cell (i.e., the cell that we should compute now) and its neighboring cells. We approximate the distance values, which should be provided by the hidden cells, by using the distance value in the current cell. Since the sampled sequences are obtained based on the sampling theorem, this is a suitable approximation.

Let X and Y be two sequences of lengths n and m with sampling periods T_x and T_y, respectively. Also let $\mathcal{X}$ $=(\mathrm{x}_1, ..., \mathrm{x}_i, ..., \mathrm{x}_{\lfloor n/T_x \rfloor})$ and $\mathcal{Y}=(\mathrm{y}_1, ..., \mathrm{y}_j, ..., \mathrm{y}_{\lfloor m/T_y \rfloor})$ be sampled sequences of X and Y, respectively. We obtain the scores of the subsequences of $\mathcal{X}$ and $\mathcal{Y}$ as follows.

$$
\begin{aligned}
&V(\mathcal{X}[i_s:i_e], \mathcal{Y}[j_s:j_e]) = v(i_e, j_e)\\
&v(i,j) = max \begin{cases} 0 \\ \varepsilon b_v - T_y \cdot ||\mathrm{x}_i - \mathrm{y}_j|| + v(i, j-1) \\ \varepsilon b_h - T_x \cdot ||\mathrm{x}_i - \mathrm{y}_j|| + v(i-1, j) \\ \varepsilon b_d - max(T_x, T_y) \cdot ||\mathrm{x}_i - \mathrm{y}_j|| + v(i-1, j-1) \end{cases}\\
&v(0,0) = v(i,0) = v(0,j) = 0\\
&(i = 1, ..., \lfloor n/T_x \rfloor;\ j = 1, ..., \lfloor m/T_y \rfloor;\\
&\ \lfloor (n-w)/T_x \rfloor \le i \le \lfloor n/Tx \rfloor;\ \lfloor (m-w)/T_y \rfloor \le j \le \lfloor m/Ty \rfloor;\,)
\end{aligned}
\tag{8}
$$

We interpolate T_y values if we select the score of the vertical cell. Similarly, we interpolate T_x values in the horizontal direction, and max(T_x, T_y) values in the diagonal direction. Furthermore, we modify the weight function for the sampling approach. For $L(l_x, l_y) = (l_x + l_y)/2$, the current L value increases by $T_y/2$ if the score of a vertical cell is chosen, by $T_x/2$ if the score of a horizontal cell is chosen, and by $(T_x + T_y)/2$ if the score of a diagonal cell is chosen. Thus, we obtain $b_v = T_y/2$, $b_h = T_x/2$ and $b_d = (T_x + T_y)/2$ for these directions.

5.2 Position matrix

The position matrix for the sampling approach is similar to Equation (7). We keep the starting position of the previous cell if any of the three neighboring cells is selected. In other cases, we appropriately set the starting position in concert with the sampling periods T_x and T_y. More specifically, the starting position $s(i, j)$ is computed as follows.

$$
s(i,j) = \begin{cases} (i \cdot T_x, j \cdot T_y) & (v(i,j) \le 0) \\ s(i, j-1) & (v(i,j-1) \ne 0 \wedge v(i,j) \\ & \quad = \varepsilon b_v - T_y \cdot ||\mathrm{x}_i - \mathrm{y}_j|| + v(i,j-1)) \\ s(i-1, j) & (v(i-1,j) \ne 0 \wedge v(i,j) \\ & \quad = \varepsilon b_h - T_x \cdot ||\mathrm{x}_i - \mathrm{y}_j|| + v(i-1,j)) \\ s(i-1, j-1) & (v(i-1,j-1) \ne 0 \wedge v(i,j) = \varepsilon b_d \\ & \quad - max(T_x, T_y) \cdot ||\mathrm{x}_i - \mathrm{y}_j|| + v(i-1,j-1)) \\ ((i-1) \cdot T_x + 1, (j-1) \cdot T_y + 1) & (otherwise). \end{cases}
\tag{9}
$$

5.3 Streaming algorithm

Algorithm 2 shows a detailed description of our sampling approach. The algorithm reflects the information about the skipped elements in the next computation and approximately computes the score and the position of the subsequence pair. The basic procedure is the same as that of the original version of CrossMatch (i.e., Algorithm 1), however, we can greatly reduce the space requirement and the computation cost by the sampling, which faithfully reconstructs the original sequence.

Lemma 5 *Let T be the sampling period. With the sampling approach, CrossMatch requires $O(w/T)$ time (per update) and space.*

Proof See appendix E. □

Algorithm 2 CrossMatch (sampling)

```
Input: new value $x_n$ at time-tick $n$
Output: (approximate) optimal subsequence pairs and DTW distances
 1: if $n \bmod T_x = 0$ then
 2:     // Detect optimal subsequence pairs.
 3:     for $j := \lfloor (m-w)/T_y \rfloor$ to $\lfloor m/T_y \rfloor$ do
 4:         $C'_v := v(n/T_x, j)$; // Score value derived by Eq. (8)
 5:         $C'_s := s(n/T_x, j)$; // Starting position derived by Eq. (9)
 6:         $C'_e := (n, j * T_y)$; // End position
 7:         if $C'_v \geq \varepsilon l_{min}$ then
 8:             // Add the subsequence pair as a new candidate.
 9:             if $C'_s \notin \mathcal{S}$ then
10:                 add $C'_v$, $C'_s$, and $C'_e$ to $\mathcal{S}$;
11:             else
12:                 for each candidate $C \in \mathcal{S}$ do
13:                     // Overwrite the maximum score.
14:                     if $C'_s = C_s \wedge C'_v \geq C_v$ then
15:                         $C_v := C'_v$;
16:                         $C_e := C'_e$;
17:                     end if
18:                 end for
19:             end if
20:         end if
21:     end for
22:     // Report the optimal subsequence pairs.
23:     for each candidate $C \in \mathcal{S}$ do
24:         if $(\forall_i, s(i, \lfloor m/T_y \rfloor) \neq C_s) \ \wedge \ (\forall_j, s(n/T_x, j) \neq C_s)$
            then
25:             $d_{min} = \varepsilon L(l_x, l_y) - C_v$;
26:             Report $d_{min}$, $C_s$ and $C_e$;
27:             Remove $C$ from $\mathcal{S}$;
28:         end if
29:     end for
30: end if
```

5.4 Adaptive sampling approach

We discussed how to compute the score assuming that the fixed sampling period of each sequence is given. Next, we focus our attention on handling the variation in the sampling period. Assuming that we cannot know the elements of a data stream in advance, the power rate between high and low frequencies might vary over time. This means that the frequency range varies locally in the time domain. We want to incorporate this frequency range variation into CrossMatch. Thus, CrossMatch updates the sampling period in stream processing. We call this method the *adaptive sampling approach* as opposed to the sampling approach.

Let $\mathcal{X} = (\mathrm{x}_1, ..., \mathrm{x}_i, ..., \mathrm{x}_{n'})$, be the sampled sequences of X, and $\mathcal{T}_x = (t_{\mathrm{x}_1}, ..., t_{\mathrm{x}_i}, ..., t_{\mathrm{x}_{n'}})$ be the sampling period of $\mathcal{X}$ in each time-tick. In the adaptive sampling approach, the number of hidden cells varies according to the sampling period in each time-tick. We compute the appropriate sampling period in each cell and approximate the distance values of the hidden cells accordingly. On the other hand, we determine the weights of each direction dynamically since the current L value is determined by the sampling period in each time-tick. For $L = (l_x + l_y)/2$, the weight of the horizontal direction in cell (i, j) is $b_h = t_{\mathrm{x}_i}/2$ and the others are similarly set by the sampling periods. In the adaptive sampling approach, we constantly use the sampling period, which reflects the sequence of recent time-ticks. Thus, this approach would be more powerful when the sequence consists of high and low frequencies.

Incremental algorithms have been proposed for computing the frequency in the stream sense (e.g., [29,49]). CrossMatch can utilize any and all of these solutions to compute the frequency efficiently. However, this research topic is beyond the scope of this paper.

Two sampling approaches do not guarantee their error-bound theoretically because the alignment of DTW depends on the data sequence and changes if the data sequence is sampled with a different sampling period. However, we show that their errors are very small in Section 7.3.

6 Discovery of group-similarity

So far, we have assumed the problem of cross-similarity between two data streams. For more generality, we would like to make CrossMatch more flexible. We now tackle a more challenging problem: how do we efficiently identify common local patterns among multiple data streams? A useful feature of CrossMatch is that it can be effectively extended to this case.

Given multiple data streams (more than two sequences), we want to find '*group-similarity*', which means the cross-similarity among them. The work in [62] has addressed the problem of similarity group-by that supports grouping based on tuples in a database. On the other hand, group-similarity provides grouping based on similar patterns. For example, in sensor networks, measurement values arriving from many different sensors have to be examined dynamically. CrossMatch makes it possible to reduce a large number of streams to just a handful of common patterns that compactly describe the key features. More importantly, the time and space requirements are constant per update.

We formally define group-similarity below. To simplify our presentation, we focus on three sequences X, Y, and Z. We first present the DTW distance for the three sequences.

Consider three sequences, $X = (x_1, x_2, ..., x_{n_1})$, $Y = (y_1, y_2, ..., y_{n_2})$, and $Z = (z_1, z_2, ..., z_{n_3})$. Their DTW distance $D(X, Y, Z)$ is defined as [2]

[2] The other settings for DTW are as follows. $d(0,0,0) = 0$.
$d(i,0,0) = d(0,j,0) = d(0,0,k) = \infty$.
$d(i,j,0) = d(0,j,k) = d(i,0,k) = \infty$.
$i = 1, ..., n_1, j = 1, ..., n_2, k = 1, ..., n_3$.
$(n_1 - w \leq i \leq n_1) \wedge (n_2 - w \leq j \leq n_2) \wedge (n_3 - w \leq k \leq n_3)$.

$$
\begin{aligned}
&D(X,Y,Z) = d(n_1,n_2,n_3)\\
&d(i,j,k) = ||x_i - y_j|| + ||y_j - z_k|| + ||z_k - x_i||\\
&\qquad + \min \begin{cases} d(i,j-1,k)\\ d(i-1,j,k)\\ d(i-1,j-1,k)\\ d(i,j,k-1)\\ d(i-1,j,k-1)\\ d(i,j-1,k-1)\\ d(i-1,j-1,k-1). \end{cases}
\end{aligned} \tag{10}
$$

The time warping matrix for the three sequences consists of $n_1 * n_2 * n_3$ cells [3]. In cell (i,j,k), we choose one cell, which has the minimum distance, from the seven neighboring cells and add the value to the distance value between three elements (x_i, y_j, and z_k). The DTW distance is obtained by accumulating these distances.

Definition 5 (Group-similarity) Given three sequences X, Y, Z, and thresholds ε and l_{min}, the subsequences $X[i_s : i_e]$, $Y[j_s : j_e]$, and $Z[k_s : k_e]$ have the property of group-similarity if they satisfy the following condition.

$$
D(X[i_s\!:\!i_e],Y[j_s\!:\!j_e],Z[k_s\!:\!k_e]) \le \varepsilon(L(l_x,l_y,l_z)-l_{min}). \tag{11}
$$

We compute the DTW distance among three sequences and detect the subsequences whose lengths are greater than l_{min}. As with cross-similarity, we face the overlap problem. The number of overlapping subsequences increases significantly with the number of sequences. We detect the best match of group-similarity as follows.

Problem 2 Given three sequences X, Y, and Z, and thresholds ε and l_{min}, we want to find subsequences $X[i_s : i_e]$, $Y[j_s : j_e]$, and $Z[k_s : k_e]$ that satisfy the following conditions.

1. $X[i_s : i_e]$, $Y[j_s : j_e]$, and $Z[k_s : k_e]$ have the property of group-similarity.
2. $D(X[i_s : i_e], Y[j_s : j_e], Z[k_s : k_e]) - \varepsilon(L(l_x, l_y, l_z) - l_{min})$ is the minimum value from a set of overlapping subsequences that satisfies the first condition.

Let $d_{i,j,k}(p,q,r)$ be the distance in cell (p,q,r) in the time warping matrix for three sequences that starts from i on the x-axis, j on the y-axis, and k on the z-axis. The distance between the subsequences of X, Y, and Z can be obtained as follows [4].

$$
\begin{aligned}
&D(X[i_s : i_e], Y[j_s : j_e], Z[k_s : k_e]) = d_{i_s,j_s,k_s}(l_x,l_y,l_z)\\
&d_{i,j,k}(p,q,r) = ||x_{i+p-1} - y_{j+q-1}|| + ||y_{j+q-1} - z_{k+r-1}||\\
&\qquad\qquad + ||z_{k+r-1} - x_{i+p-1}|| + d_{min}
\end{aligned} \tag{12}
$$

Note that d_{min} is the minimum distance between the seven neighboring cells.

Lemma 6 *The naive solution requires $O(n_1w^3 + n_2w^3 + n_3w^3)$ time (per update) and space to discover the group-similarity for three sequences.*

Proof See appendix F. □

How do we detect group-similarity with CrossMatch? We need only two matrices for computation (i.e., the score and position matrices). Each matrix has only two planes (i.e., previous and current planes) for each sequence X, Y, and Z. For each incoming data point, we calculate $O(w^2)$ score values and update $O(w^2)$ starting positions. Specifically, given three sequences X, Y, and Z and warping scope w, the score $V(X[i_s\,\!:\!!i_e], Y[j_s\!:\!j_e], Z[k_s\!:\!k_e])$ of $X[i_s\!:\!i_e]$, $Y[j_s\!:\!j_e]$ and $Z[k_s\!:\!k_e]$ is computed as follows [5].

$$
\begin{aligned}
&V(X[i_s : i_e], Y[j_s : j_e], Z[k_s : k_e]) = v(i_e, j_e, k_e)\\
&v(i,j,k) = \max(0, v_{best})
\end{aligned} \tag{13}
$$

For cell (i,j,k), we choose one of seven neighboring cells and determine the score if the score v_{best} is not a negative value. For example, if the score of a diagonal cell $(i-1, j-1, k-1)$ is chosen, we have

$$
\begin{aligned}
&v_{best} = \varepsilon b_d - d_{cell} + v(i-1, j-1, k-1)\\
&d_{cell} = ||x_i - y_j|| + ||y_j - z_k|| + ||z_k - x_i||
\end{aligned}
$$

where d_{cell} is the distance between the cells of the three sequences. While computing the score values, we update the starting position in the position matrix. If we choose one of the seven neighboring cells, we keep the same starting position. If not, we choose the current cell (i,j,k) as the starting position. Thus, we can deal with the score computation and the updating of the starting position very effectively.

Lemma 7 *Given three sequences X, Y, and Z, Problem 2 is equivalent to the following conditions.*

1. $V(X[i_s\!:\!i_e], Y[j_s\!:\!j_e], Z[k_s\!:\!k_e]) \ge \varepsilon l_{min}$

[3] Here we focus on a third-order tensor for time warping, namely the time warping tensor for three sequences. However, for simplicity, we shall use the term "time warping matrix" in this paper.

[4] The other settings are as follows. $d_{i,j,k}(0,0,0) = 0$, $d_{i,j,k}(p,0.0) = d_{i,j,k}(0,q,0) = d_{i,j,k}(0,0,r) = \infty$.
$d_{i,j,k}(p,q,0) = d_{i,j,k}(0,q,r) = d_{i,j,k}(p,0,r) = \infty$.
$i = 1,...,n_1, p = 1,...,n_1-i+1, j = 1,...,n_2, q = 1,...,n_2-j+1,$
$k = 1,...,n_3, r = 1,...,n_3-k+1.$
$(n_1-w \le i+p \le n_1) \wedge (n_2-w \le j+q \le n_2) \wedge (n_3-w \le k+r \le n_3).$

[5] The other settings for the scoring function are as follows.
$v(0,0,0) = v(i,0,0) = v(0,j,0) = v(0,0,k) = 0.$
$v(i,j,0) = v(0,j,k) = v(i,0,k) = 0.$
$(n_1-w \le i \le n_1) \wedge (n_2-w \le j \le n_2) \wedge (n_3-w \le k \le n_3).$

2. *$V(X[i_s : i_e], Y[j_s : j_e], Z[k_s : k_e]) - \varepsilon l_{min}$ is the maximum value in each group of subsequences that the warping path crosses.*

Proof See appendix G. □

From Lemma 7, Equation (11) holds for any reported subsequences. As with Lemma 2, it is obvious that CrossMatch reports the optimal subsequences from the set of overlapping subsequences. Thus, CrossMatch guarantees the correctness of the result for group-similarity.

Lemma 8 *CrossMatch requires $O(w^2)$ time (per update) and $O(w^2)$ space to discover the group-similarity for three sequences.*

Proof See appendix H. □

Although the complexity of group-similarity is still quadratic with respect to the number of sequences, CrossMatch is much faster in practice than the naive solution and enables the examination of very large collections of sequences.

7 Experimental Evaluation

We performed experiments to evaluate the effectiveness of CrossMatch. Our experiments were conducted on a 2.4-GHz Intel Core 2 machine with 4 GB of memory, running Linux. The experiments were designed to answer the following questions.

1. How well does CrossMatch provide the optimal subsequences without redundant information?
2. How successful is CrossMatch in capturing cross-similarity?
3. How effective is the sampling approach in capturing cross-similarity?
4. How well does CrossMatch scale with the sequence length in terms of computation time and memory space?
5. How well does CrossMatch work in high-dimensional data streams?
6. How well does CrossMatch identify group-similarity?

We used real and synthetic datasets for the experiments. These datasets (except high-dimensional sequences) are available for downloading from the web page [6]. The details of each dataset are provided in the following subsections.

[6] http://www.kecl.ntt.co.jp/icl/ls/members/machiko/time-series.zip

7.1 Filtering redundant information

We compared CrossMatch with the previous algorithm [66] to investigate its effectiveness in filtering redundant information [7]. We used a synthetic dataset, *Sines*, which consists of discontinuous sine waves with white noise (see Fig. 8 (a)), and for our previous algorithm and CrossMatch we set l_{min} at 15% of the sequence length, ε at 1.0e-2, and w at 50% of the sequence length.

Fig. 8 (b) plots the sequence length vs. the number of detected subsequence pairs for the two algorithms. In the previous algorithm, increases in sequence length trigger a large increase in the number of detected subsequence pairs. CrossMatch, on the other hand, detects fewer subsequence pairs than the previous algorithm.

Fig. 8 (c) shows how CrossMatch captures subsequence pairs. For visualization purposes, we find the optimal warping path by backtracking the selected cells from the end position, and plot the cells from (i_e, j_e) to (i_s, j_s) for the subsequence pair $X[i_s : i_e]$ and $Y[j_s : j_e]$. Unlike the previous algorithm, CrossMatch provides only the optimal subsequence pairs in a streaming fashion. Therefore, by eliminating the overlapping subsequence pairs, the periodicity of cross-similarity is revealed and users can obtain ‘real’ results without receiving redundant information.

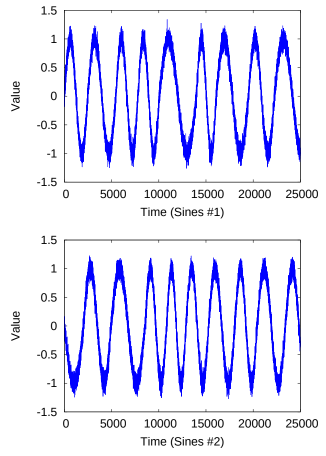
7.2 Detecting cross-similarity between two sequences

We present case studies of real and synthetic datasets to demonstrate the effectiveness of our approach in discovering optimal subsequence pairs. We set l_{min} at 500 for *RandomSines* and at 1000 for *Spikes* in each synthetic dataset. We set l_{min} at 15% of the sequence length for real datasets (i.e., *Humid*, *Automobile traffic*, *Web*, *Sunspots*, and *Temperature*). The warping scope w was set at 50% of the sequence length for all datasets. The details of each dataset and the settings for the experiments are given in Table 3. In Fig. 10, the left and center figures represent the datasets and the right figures represent the optimal warping paths of cross-similarity detected from these datasets.

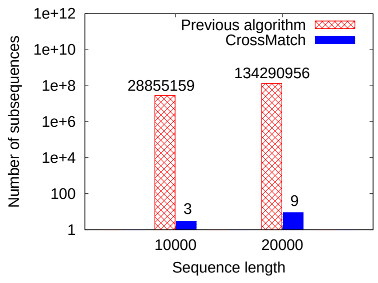
7.2.1 RandomSines

We used a synthetic dataset, *RandomSines*, which consists of discontinuous sine waves with white noise (see Fig. 9 (a)). This dataset includes different-length intervals between the sine waves, which were generated using a random walk function. We varied the period of each sine wave and the intervals between these sine waves in the sequence.

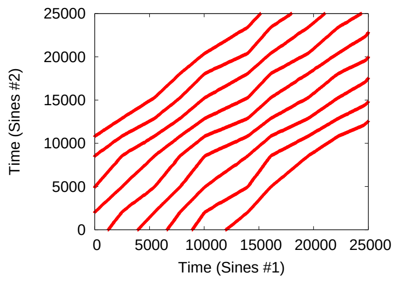
[7] The previous algorithm does not introduce the warping scope w. To ensure the validity of the experiment, we modified the algorithm and introduced the warping scope.



(a) Data sequences



(b) Number of detected subsequence pairs



(c) Discovery of cross-similarity with CrossMatch

Fig. 8 Effect of filtering redundant information. CrossMatch correctly outputs the best match of cross-similarity without redundant information.

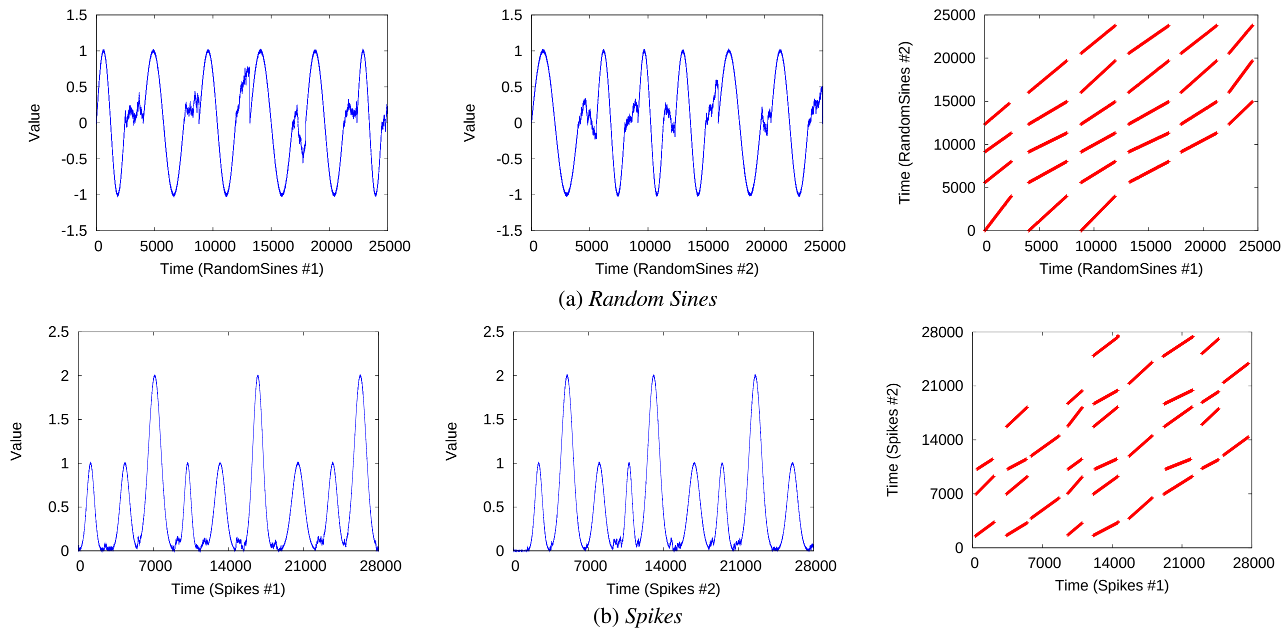


Fig. 9 Discovery of cross-similarity using *Random Sines* and *Spikes*. The left and center figures represent data sequences and the right figures represent the optimal warping paths of cross-similarity.

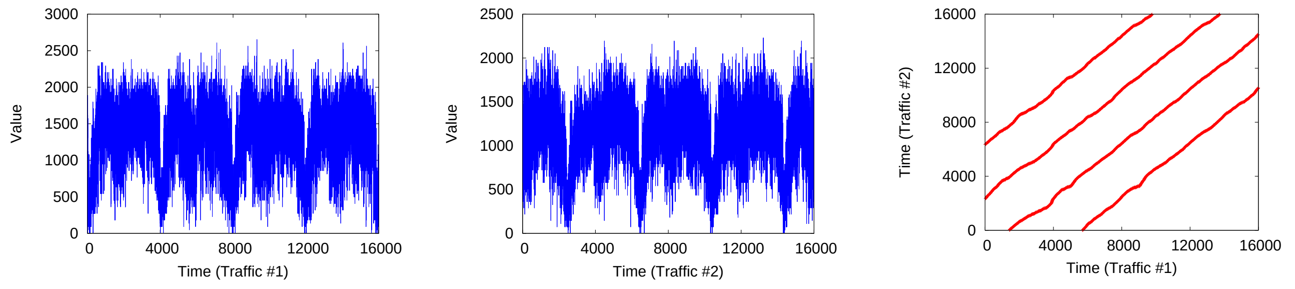
As shown in the right figure of Fig. 9 (a), CrossMatch perfectly identifies all the sine waves and their time-varying periodicities. In this figure, the difference in the period of each sine wave appears as a difference in the slope.

7.2.2 Spikes

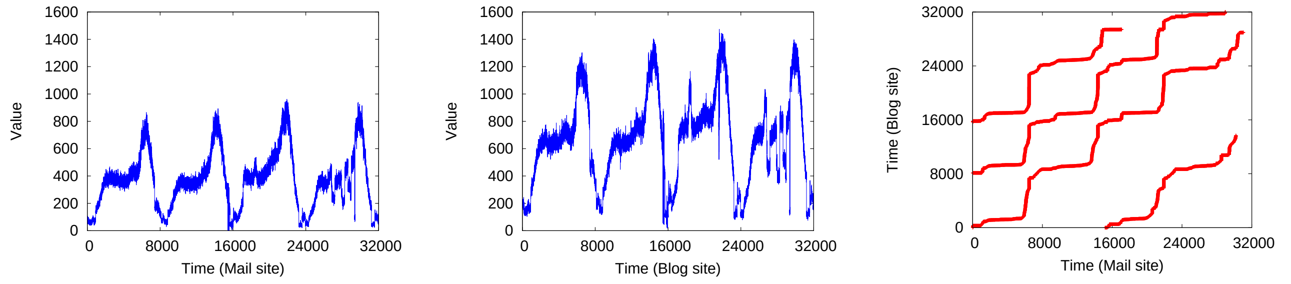
This is the synthetic dataset shown in Fig. 9 (b), which consists of large and small spikes. The data for different-length intervals between spikes were generated using a random walk function. The period of each spike is also different. As seen in the right figure of Fig. 9 (b), we confirm that CrossMatch detects both large and small spikes. The difference in the period of each spike appears as a difference in plot length; wide spikes indicate long plot lengths and narrow spikes indicate short plot lengths.

7.2.3 Humidity

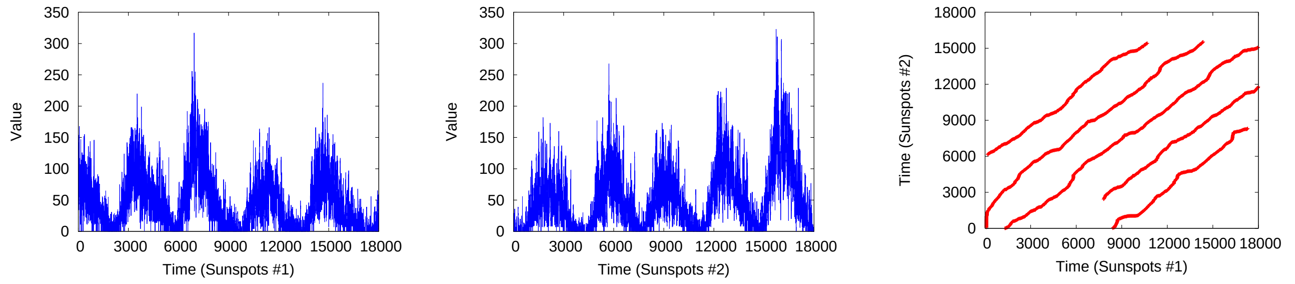
Fig. 1 shows the detected subsequence pairs for the humidity dataset. CrossMatch captures common patterns except for the dissimilar sections. Our method is designed to find the similar subsequence pairs. However, by applying it to



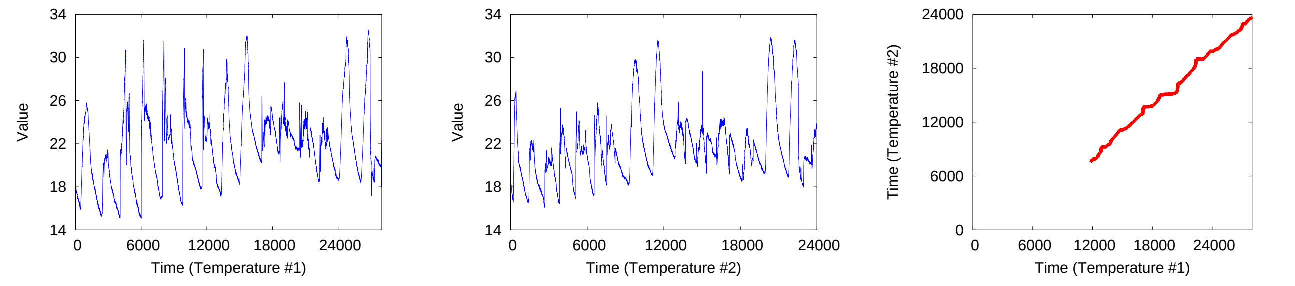
(a) *Automobile traffic*



(b) *Web*



(c) *Sunspots*



(d) *Temperature*

Fig. 10 Discovery of cross-similarity using *Automobile traffic*, *Web*, *Sunspots*, and *Temperature*.

sequences that are roughly similar, it can utilize the discovery of dissimilar sections.

7.2.4 Automobile traffic

Fig. 10 (a) shows time-series data of automobile traffic, which has a daily period. Each day contains other distinct patterns for the morning and afternoon rush hours. Hourly traffic is bursty data, and we can regard it as white noise.

CrossMatch is successful in accurately detecting the daily period without being deceived by the high-frequency hourly traffic. Consecutive lines and their regular intervals indicate periodicity. Moreover, the intervals between the consecutive lines correspond to the daily period, and we can confirm that the characteristics of the data are revealed by the cross-similarity thus detected.

7.2.5 Web

Fig. 10 (b) shows access counts for mail and blog sites obtained every 10 seconds. We observe the daily periodicity of sequences, which increases from morning to night and reaches a peak.

Table 3 Details of datasets and parameter settings.

Datasets	Sequence length		ε
	Seq. #1	Seq. #2	
RandomSines	25000	25000	1.0e-4
Spikes	28000	28000	5.0e-6
Humidity	26779	40831	8.0e-1
Automobile traffic	16000	16000	8.5e+4
Web	32000	32000	4.0e+4
Sunspots	18000	18000	3.0e+2
Temperature	28000	24000	2.0e-1

The right figure in Fig. 10 (b) confirms that CrossMatch identified the periodicity. The figure shows winding lines, unlike *Automobile*. This indicates that CrossMatch aligned the elements of sequences that were stretched along the time axis. Cross-similarity is detected by the time-scaling property of CrossMatch.

7.2.6 Sunspots

Fig. 10 (c) is sunspots dataset recorded on a daily basis. This is a well-known dataset whose time-varying periodicity is related to sun activity. The average number of visible sunspots increases when the sun is active and decreases when the sun is inactive. This change occurs with a regular period of about 11 years.

CrossMatch distinguishes the increase and decrease in the average number and captures similar periods.

7.2.7 Temperature

We use the temperature measurements (degrees Celsius) from the Critter dataset, which are obtained with small sensors (see Fig. 10 (d)). The sensors give one reading approximately every minute. This dataset has many missing values and the lengths of the two sequences are different. These sequences consist of similar changes with a temperature fluctuation of 18 to 32 °C.

Despite the missing measurement values and the difference in the period, CrossMatch successfully detected the pattern.

7.3 Effect of sampling approach

In this section, we show the results we obtained with the sampling and adaptive sampling approaches. We used four real datasets for the experiment. To determine the sampling period for each dataset in the sampling approach, we computed a power spectrum from the normalized sequences. Real datasets often include high frequencies with very low energy. The Nyquist frequencies for such datasets could be extremely high. Since the frequency limit is widely understood in various fields (e.g., audio processing and network analysis), in settings regarding Nyquist frequency, we disregard the high frequency components, whose power is very small. We set the power value threshold at 5.0e-4 in this experiment. The main energy of *Traffic #1* is distributed in the $0 \le f \le 1478$ frequency range and the Nyquist frequency is $f_{N_q} = 2596/n$. Similarly, the main energy of *Traffic #2* is distributed in the $0 \le f \le 1438$ frequency range and the Nyquist frequency is $f_{N_q} = 2876/m$. Thus, the sampling periods are $T_x = 5$ and $T_y = 6$ [8] . With the adaptive sampling approach, the sampling rate varies depending on the frequency range.

Fig. 11 presents the cross-similarity with the sampling approach. We omit the results we obtained with the adaptive sampling approach because they are almost the same as those of the sampling approach. The optimal subsequence pairs as well as those estimated in Fig. 10 provide an accurate assessment of the similarity between sequences. The original and sampling approaches offer very similar cross-similarity and are equally useful.

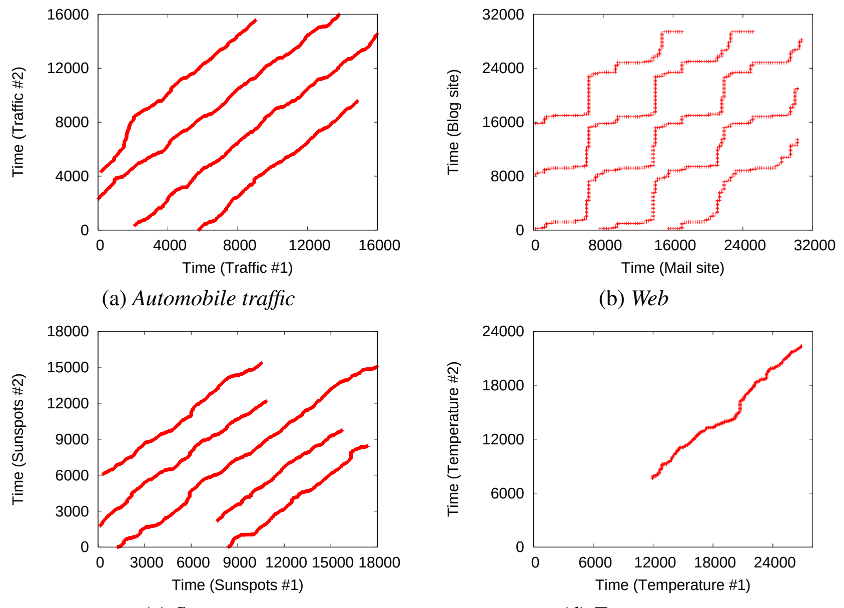
Next, we show the correctness of the results obtained with the two sampling approaches. We used the Diarization Error Rate (DER) [26] to evaluate the accuracy. DER is a primary metric for speech recognition and presents the ratio of incorrect speech time against the total amount of exact speech time. In our evaluation, DER is represented by the ratio of the mismatch length to the exact subsequence length and is defined as follows.

$$DER_x = \frac{|i_s - i'_s| + |i_e - i'_e|}{i_e - i_s + 1}$$
$$DER_y = \frac{|j_s - j'_s| + |j_e - j'_e|}{j_e - j_s + 1}$$

Note that (i_s, j_s) is the exact starting position for the original CrossMatch and (i'_s, j'_s) is the approximate starting position for the two sampling approaches. The end positions are also the same. We calculate the DER for every detected subsequence pair and show their average values.

Table 4 gives the DER results. Although there are variabilities in each dataset, the two sampling approaches closely identify the positions of optimal subsequence pairs. In particular, the DERs of the two approaches are different for *Temperature*. As compared with other datasets, which consist of high frequency, *Temperature* contains both of high and low frequencies. Therefore, it is considered that the algorithm approximates the score in response to the variation of the sampling period. As we expected, the adaptive

[8] The frequency components f and the sampling periods T of the other datasets are as follows.
Mail: $0 \le f \le 63$, $T_x = 254$. *Blog*: $0 \le f \le 80$, $T_y = 200$.
Sunspots #1: $0 \le f \le 1431$, $T_x = 6$. *#2*: $0 \le f \le 1497$, $T_y = 6$.
Temperature #1: $0 \le f \le 201$, $T_x = 70$. *#2*: $0 \le f \le 137$, $T_y = 88$.



(a) *Automobile traffic* (b) *Web*

(c) *Sunspots* (d) *Temperature*

Fig. 11 Discovery of cross-similarity with the sampling approach. The sampling approach correctly identifies the patterns as effectively as the original approach.

Table 4 Diarization error rate in two sampling approaches.

	sampling approach		adaptive sampling	
	DER_x	DER_y	DER_x	DER_y
Automobile traffic	0.066	0.094	0.074	0.095
Web	0.039	0.042	0.039	0.036
Sunspots	0.105	0.121	0.093	0.106
Temperature	0.068	0.084	0.002	0.015

sampling approach has an advantage as regards datasets that consist of high and low frequencies.

7.4 Performance

We compared two CrossMatch approaches (i.e., the original and sampling approaches [9]) with the naive solution and an existing method, SPRING, in terms of computation time and memory space. SPRING detects high-similarity subsequences that are similar to a fixed-length query sequence under the DTW distance [58]. SPRING is not intended to be used for finding cross-similarity, but we can apply this method to evaluate the efficiency and to verify the complexity of CrossMatch. SPRING requires $O(n+m)$ matrices, thus an algorithm with SPRING for finding cross-similarity requires $O(nw+mw)$ time (per update) and space. We used *Temperature* for this experiment.

Fig. 12 shows the experimental results with regard to computation time. This is the average processing time per time-tick for each sequence length. As we expected, CrossMatch identifies the optimal subsequence pairs much faster than the naive and SPRING implementations. The trend shown in this figure agrees with our theoretical discussion in Section 4.3.2. In particular, the sampling approach significantly reduces the computation time.

Fig. 13 compares Naive, SPRING, and the two CrossMatch approaches in terms of memory space. The x-axis represents data stream length. The space requirement of CrossMatch is clearly lower than those of the Naive and SPRING implementations, and the sampling approach can greatly reduce the space requirement. The results also agree with our theoretical discussion in Section 4.3.2.

7.5 Extension to high-dimensional data streams

We applied CrossMatch to high-dimensional data streams. One interesting problem is how to apply CrossMatch to motion capture data (see tables in Fig. 14). *Mocap* is a real dataset created by recording motion information from a human actor while the actor performed different actions (e.g.,

[9] We show only the result for the sampling approach since the average sampling periods were almost the same between two approaches.

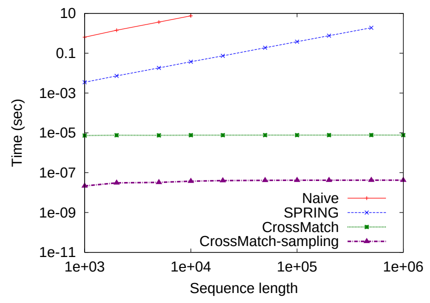


Fig. 12 Wall clock time per time-tick as a function of sequence length. CrossMatch provides significantly better performance.

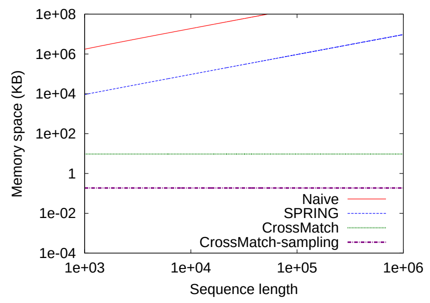


Fig. 13 Memory space consumption as a function of sequence length. CrossMatch detects cross-similarity with a constant memory space.

Time-tick	Motions
1 - 762	walking
763 - 1766	kicking
1767 - 2087	jumping upward
2088 - 2945	walking
2946 - 3444	running
3445 - 4144	jumping forward
4145 - 5007	walking

Mocap #1

Time-tick	Motions
1 - 813	walking
814 - 1278	jumping upward
1279 - 2091	punching
2092 - 3093	walking
3094 - 3660	twist dancing
3661 - 5042	golf swing

Mocap #2

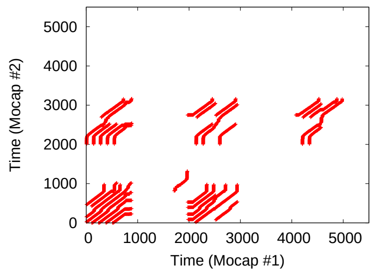


Fig. 14 Discovery of cross-similarity in *Mocap*. CrossMatch successfully detects the same motions (i.e., walking and jumping upward) in multi-dimensional sequences.

walking, running, and kicking). Special markers are placed on the actor's joints (e.g., knees and elbows), and their x-, y- and z-velocities are recorded at about 120 Hz.

X and Y are multi-dimensional time sequences, and our goal is to find matching subsequences between X and Y. Intuitively, if X and Y include the same motions, we want to find these motions.

We used sequences obtained from the CMU motion capture database [10]. We selected the data for limbs from the original data and used them as 8-dimensional data. Each motion is listed in the tables in Fig. 14. The dataset has two motions in common (i.e., walking and jumping upward), and the length of each motion is different. We set l_{min} at 240, which corresponds to about two seconds, ε at 10, and w at 50% of the sequence length.

The result shown in Fig. 14 reveals that CrossMatch can accurately capture the two motions. We can confirm that the walking motion yields high cross-similarity. There are many shifted sequences, because walking is a repetitive behavior in which the limbs move back and forth. CrossMatch works for high-dimensional datasets and detects the repetitive motion as cross-similarity.

[10] http://mocap.cs.cmu.edu/

7.6 Detecting group-similarity

We performed an experiment to discover group-similarity. We used the *Sensor* dataset for this experiment (see Fig. 15). *Sensor* consists of three streams that represent temperature readings from sensors within several buildings. Each sensor provides a reading every 4 minutes. Overall, the dataset fluctuates greatly at different time-ticks but the three sensors exhibit a similar fluctuation pattern.

We show the experimental result in the right figure of Fig. 15. The optimal warping path is plotted as a line in 3-dimensional space. CrossMatch discovers the fluctuation pattern in spite of the difference in the periodicity. In multiple sequences, providing a concise summary of key trends is a significant challenge. CrossMatch summarizes the three sequences into a manageable synoptic pattern and captures the characteristics shared by the sequences.

8 Conclusions

We described the problem of finding common local patterns based on DTW over data streams and presented a practical solution. CrossMatch is a one-pass algorithm based on DTW, which detects local common patterns in constant time (per update) and space without sacrificing accuracy. A theoretical analysis and experiments demonstrated that CrossMatch works as expected. Furthermore, we have provided an enhancement, a sampling approach, to greatly compress the size of the matrices. We showed that the sampling approach further improves the efficiency of CrossMatch in terms of time and space requirements.

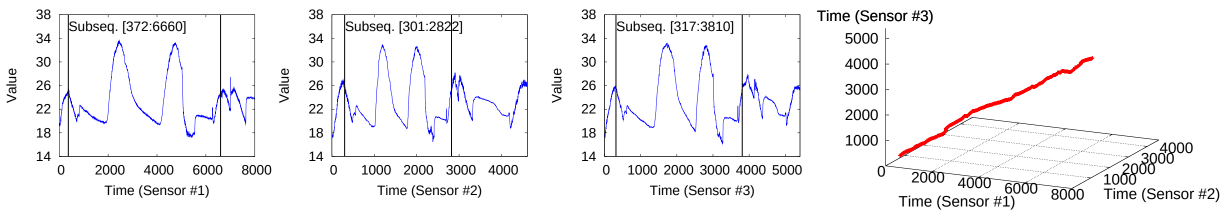


Fig. 15 Discovery of group-similarity using *Sensor*. CrossMatch captures fluctuation pattern correctly.

CrossMatch has the following characteristics.

- In contrast to the naive solution, CrossMatch greatly improves performance and can be processed at a high speed.
- CrossMatch requires constant space (per update) to detect cross-similarity or group-similarity, and it consumes only a small quantity of resources.
- Despite the high-speed processing, CrossMatch guarantees correct results.
- CrossMatch works efficiently for high-dimensional data streams.

As a result, our detailed study provides many insights into the applicability and use of CrossMatch. In particular, CrossMatch proved crucial in producing significantly more concise and informative patterns, without any prior knowledge about the data.

References

1. Abadi, D.J., Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S.B.: Aurora: a new model and architecture for data stream management. VLDB J. **12**(2), 120–139 (2003)
2. Agrawal, R., Faloutsos, C., Swami, A.N.: Efficient similarity search in sequence databases. In: FODO, pp. 69–84 (1993)
3. Arasu, A., Babcock, B., Babu, S., McAlister, J., Widom, J.: Characterizing memory requirements for queries over continuous data streams. In: PODS, pp. 221–232 (2002)
4. Assent, I., Wichterich, M., Krieger, R., Kremer, H., Seidl, T.: Anticipatory dtw for efficient similarity search in time series databases. PVLDB **2**(1), 826–837 (2009)
5. Athitsos, V., Papapetrou, P., Potamias, M., Kollios, G., Gunopulos, D.: Approximate embedding-based subsequence matching of time series. In: SIGMOD Conference, pp. 365–378 (2008)
6. Babcock, B., Babu, S., Datar, M., Motwani, R.: Chain : Operator scheduling for memory minimization in data stream systems. In: SIGMOD Conference, pp. 253–264. San Diego, California (2003)
7. Babcock, B., Datar, M., Motwani, R.: Sampling from a moving window over streaming data. In: SODA, pp. 633–634. San Francisco, CA (2002)
8. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The r*-tree: An efficient and robust access method for points and rectangles. In: SIGMOD Conference (1990)
9. Berndt, D.J., Clifford, J.: Finding patterns in time series: A dynamic programming approach. In: Advances in Knowledge Discovery and Data Mining, pp. 229–248 (1996)
10. Cao, H., Wolfson, O., Trajcevski, G.: Spatio-temporal data reduction with deterministic error bounds. VLDB J. **15**(3), 211–228 (2006)
11. Carney, D., Cetintemel, U., Rasin, A., Zdonik, S.B., Cherniack, M., Stonebraker, M.: Operator scheduling in a data stream manager. In: VLDB, pp. 838–849. Berlin, Germany (2003)
12. Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., Shah, M.A.: Telegraphcq: Continuous dataflow processing for an uncertain world. In: CIDR. Asilomar (2003)
13. Chandrasekaran, S., Franklin, M.J.: Remembrance of streams past: Overload-sensitive management of archived streams. In: VLDB, pp. 348–359 (2004)
14. Chen, L., Ng, R.T.: On the marriage of lp-norms and edit distance. In: VLDB, pp. 792–803 (2004)
15. Chen, L., Özsu, M.T., Oria, V.: Robust and fast similarity search for moving object trajectories. In: SIGMOD Conference, pp. 491–502 (2005)
16. Chen, Y., Nascimento, M.A., Ooi, B.C., Tung, A.K.H.: Spade: On shape-based pattern detection in streaming time series. In: ICDE, pp. 786–795 (2007)
17. Cormode, G., Datar, M., Indyk, P., Muthukrishnan, S.: Comparing data streams using hamming norms (how to zero in). In: VLDB, pp. 335–345. Hong Kong, China (2002)
18. Cormode, G., Korn, F., Tirthapura, S.: Time-decaying aggregates in out-of-order streams. In: PODS, pp. 89–98 (2008)
19. Cranor, C.D., Johnson, T., Spatscheck, O., Shkapenyuk, V.: Gigascope: A stream database for network applications. In: SIGMOD Conference, pp. 647–651. San Diego, California (2003)
20. Das, A., Gehrke, J., Riedewald, M.: Approximate join processing over data streams. In: SIGMOD Conference, pp. 40–51. San Diego, California (2003)
21. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. In: SODA, pp. 635–644. San Francisco, CA (2002)
22. Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E.J.: Querying and mining of time series data: experimental comparison of representations and distance measures. PVLDB **1**(2), 1542–1552 (2008)
23. Dobra, A., Garofalakis, M.N., Gehrke, J., Rastogi, R.: Processing complex aggregate queries over data streams. In: SIGMOD Conference, pp. 61–72. Madison, Wisconsin (2002)
24. Eckmann, J.P., Kamphorst, S.O., Ruelle, D.: Recurrence plots of dynamical systems. Europhysics Letters (epl) **4**, 973–977 (1987)
25. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast subsequence matching in time-series databases. In: SIGMOD Conference, pp. 419–429 (1994)
26. Fiscus, J.G., Ajot, J., Garofolo, J.S.: The rich transcription 2007 meeting recognition evaluation. In: Multimodal Technologies for Perception of Humans, International Evaluation Workshops CLEAR 2007 and RT 2007, *Lecture Notes in Computer Science*, vol. 4625, pp. 373–389. Springer (2008)

27. Ganguly, S., Garofalakis, M.N., Rastogi, R.: Processing set expressions over continuous update streams. In: SIGMOD Conference, pp. 265–276. San Diego, California (2003)
28. Gehrke, J.E., Korn, F., Srivastava, D.: On computing correlated aggregates over continual data streams. In: SIGMOD Conference. Santa Barbara, California (2001)
29. Gilbert, A.C., Guha, S., Indyk, P., Muthukrishnan, S., Strauss, M.: Near-optimal sparse fourier representations via sampling. In: STOC, pp. 152–161 (2002)
30. Gilbert, A.C., Kotidis, Y., Muthukrishnan, S., Strauss, M.: Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In: VLDB, pp. 79–88. Rome, Italy (2001)
31. Han, W.S., Lee, J., Moon, Y.S., Jiang, H.: Ranked subsequence matching in time-series databases. In: VLDB, pp. 423–434 (2007)
32. Jang, J.S.R., Lee, H.R.: Hierarchical filtering method for content-based music retrieval via acoustic input. In: ACM Multimedia, pp. 401–410 (2001)
33. Kanth, K.V.R., Agrawal, D., Singh, A.K.: Dimensionality reduction for similarity searching in dynamic databases. In: SIGMOD Conference, pp. 166–176 (1998)
34. Kawasaki, H., Yatabe, T., Ikeuchi, K., Sakauchi, M.: Automatic modeling of a 3d city map from real-world video. In: ACM Multimedia, vol. 1, pp. 11–18 (1999)
35. Keogh, E.J.: Exact indexing of dynamic time warping. In: VLDB, pp. 406–417 (2002)
36. Keogh, E.J., Chakrabarti, K., Mehrotra, S., Pazzani, M.J.: Locally adaptive dimensionality reduction for indexing large time series databases. In: SIGMOD Conference, pp. 151–162 (2001)
37. Korn, F., Muthukrishnan, S., Wu, Y.: Modeling skew in data streams. In: SIGMOD Conference, pp. 181–192 (2006)
38. Lange, R., Farrell, T., Dürr, F., Rothermel, K.: Remote real-time trajectory simplification. In: PerCom, pp. 1–10 (2009)
39. Lathi, B.P.: Signal Processing and Linear Systems. Oxford University Press (1998)
40. Li, L., McCann, J., Pollard, N.S., Faloutsos, C.: Dynammo: mining and summarization of coevolving sequences with missing values. In: KDD, pp. 507–516 (2009)
41. Madden, S., Shah, M.A., Hellerstein, J.M., Raman, V.: Continuously adaptive continuous queries over streams. In: SIGMOD Conference, pp. 49–60. Madison, Wisconsin (2002)
42. Matsubara, Y., Sakurai, Y., Faloutsos, C., Iwata, T., Yoshikawa, M.: Fast mining and forecasting of complex time-stamped events. In: KDD, pp. 271–279 (2012)
43. Mokbel, M.F., Aref, W.G.: SOLE: scalable on-line execution of continuous queries on spatio-temporal data streams. VLDB J. **17**(5), 971–995 (2008)
44. Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G.S., Olston, C., Rosenstein, J., Varma, R.: Query processing, approximation, and resource management in a data stream management system. In: CIDR. Asilomar (2003)
45. Mount, D.W.: Bioinformatics: Sequence and Genome Analysis. Cold Spring Harbor Laboratory (2000)
46. Mueen, A., Keogh, E.J.: Online discovery and maintenance of time series motifs. In: KDD, pp. 1089–1098 (2010)
47. Mueen, A., Nath, S., Liu, J.: Fast approximate correlation for massive time-series data. In: SIGMOD Conference, pp. 171–182 (2010)
48. Nasraoui, O., Rojas, C., Cardona, C.: A framework for mining evolving trends in web data streams using dynamic learning and retrospective validation. Computer Networks **50**(10), 1488–1512 (2006)
49. Ogras, Ü.Y., Ferhatosmanogl, H.: Online summarization of dynamic time series data. VLDB J. (2006)
50. Papadimitriou, S., Brockwell, A., Faloutsos, C.: Adaptive, hands-off stream mining. In: VLDB, pp. 560–571 (2003)
51. Papadimitriou, S., Sun, J., Faloutsos, C.: Streaming pattern discovery in multiple time-series. In: VLDB, pp. 697–708 (2005)
52. Papadimitriou, S., Yu, P.S.: Optimal multi-scale patterns in time series streams. In: SIGMOD Conference, pp. 647–658 (2006)
53. Popivanov, I., Miller, R.J.: Similarity search over time-series data using wavelets. In: ICDE, pp. 212–224 (2002)
54. Rabiner, L., Juang, B.H.: Fundamentals of Speech Recognition. Prentice Hall, Englewood Cliffs, NJ (1993)
55. Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., Keogh, E.: Searching and mining trillions of time series subsequences under dynamic time warping. In: KDD, pp. 262–270 (2012)
56. Reeves, G., Liu, J., Nath, S., Zhao, F.: Managing massive time series streams with multi-scale compressed trickles. PVLDB **2**(1), 97–108 (2009)
57. Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. IEEE Transactions on Acoustics, Speech and Signal Processing **26**(1), 43–49 (1978)
58. Sakurai, Y., Faloutsos, C., Yamamuro, M.: Stream monitoring under the time warping distance. In: ICDE, pp. 1046–1055 (2007)
59. Sakurai, Y., Li, L., Matsubara, Y., Faloutsos, C.: WindMine: Fast and effective mining of web-click sequences. In: SDM, pp. 759–770 (2011)
60. Sakurai, Y., Papadimitriou, S., Faloutsos, C.: BRAID: stream mining through group lag correlations. In: SIGMOD Conference, pp. 599–610 (2005)
61. Sakurai, Y., Yoshikawa, M., Faloutsos, C.: FTW: fast similarity search under the time warping distance. In: PODS, pp. 326–337 (2005)
62. Silva, Y.N., Aref, W.G., Ali, M.H.: Similarity group-by. In: ICDE, pp. 904–915 (2009)
63. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of molecular biology **147**, 195–197 (1981)
64. Tatbul, N., Cetintemel, U., Zdonik, S.B., Cherniack, M., Stonebraker, M.: Load shedding in a data stream manager. In: VLDB, pp. 309–320. Berlin, Germany (2003)
65. Toyoda, M., Sakurai, Y.: Discovery of cross-similarity in data streams. In: ICDE, pp. 101–104 (2010)
66. Toyoda, M., Sakurai, Y., Ichikawa, T.: Identifying similar subsequences in data streams. In: DEXA, *Lecture Notes in Computer Science*, vol. 5181, pp. 210–224 (2008)
67. Vlachos, M., Gunopulos, D., Kollios, G.: Discovering similar multidimensional trajectories. In: ICDE, pp. 673–684 (2002)
68. Wu, H., Salzberg, B., Zhang, D.: Online event-driven subsequence matching over financial data streams. In: SIGMOD Conference, pp. 23–34 (2004)
69. Yankov, D., Keogh, E., Lonardi, S., Fu, A.W.C.: Dot plots for time series analysis. In: ICTAI, pp. 159–168 (2005)
70. Yi, B.K., Faloutsos, C.: Fast time sequence indexing for arbitrary lp norms. In: VLDB, pp. 385–394 (2000)
71. Yi, B.K., Jagadish, H.V., Faloutsos, C.: Efficient retrieval of similar time sequences under time warping. In: ICDE, pp. 201–208 (1998)
72. Zhou, M., Wong, M.H.: Efficient online subsequence searching in data streams under dynamic time warping distance. In: ICDE, pp. 686–695 (2008)
73. Zhu, Y., Shasha, D.: Statstream: Statistical monitoring of thousands of data streams in real time. In: VLDB, pp. 358–369 (2002)
74. Zhu, Y., Shasha, D.: Efficient elastic burst detection in data streams. In: KDD, pp. 336–345 (2003)
75. Zhu, Y., Shasha, D.: Warping indexes with envelope transforms for query by humming. In: SIGMOD Conference, pp. 181–192 (2003)

Appendix

A Proof of Lemma 1

We assume that the warping path from cells (i_s, j_s) through (i_e, j_e) includes cell $(i_e, j_e - 1)$ in the time warping matrix starting from (i_s, j_s) and the score matrix. From Equation (4), we have

$$||x_{i_e} - y_{j_e}|| = d_{i_s,j_s}(l_x, l_y) - d_{i_s,j_s}(l_x, l_y - 1).$$

From $b_v = L(l_x, l_y) - L(l_x, l_y - 1)$ in Equation (5), we have

$$\begin{aligned}||x_{i_e} - y_{j_e}|| &= \varepsilon L(l_x, l_y) - v(i_e, j_e)\\ &\quad -\varepsilon L(l_x, l_y - 1) + v(i_e, j_e - 1).\end{aligned}$$

Similarly, if the warping path includes cell $(i_e - 1, j_e)$ or $(i_e - 1, j_e - 1)$, we have

$$\begin{aligned}||x_{i_e} - y_{j_e}|| &= d_{i_s,j_s}(l_x, l_y) - d_{i_s,j_s}(l_x - 1, l_y)\\ &= \varepsilon L(l_x, l_y) - v(i_e, j_e)\\ &\quad -\varepsilon L(l_x - 1, l_y) + v(i_e - 1, j_e).\\ ||x_{i_e} - y_{j_e}|| &= d_{i_s,j_s}(l_x, l_y) - d_{i_s,j_s}(l_x - 1, l_y - 1)\\ &= \varepsilon L(l_x, l_y) - v(i_e, j_e)\\ &\quad -\varepsilon L(l_x - 1, l_y - 1) + v(i_e - 1, j_e - 1).\end{aligned}$$

From

$$||x_{i_s} - y_{j_s}|| = d_{i_s,j_s}(1, 1) = \varepsilon - v(i_s, j_s),$$

the time warping and the score matrices, which have the same starting position (i_s, j_s), share the same warping path since $||x_i - y_j||$ is equal in all corresponding cells. The sum of the weights on the warping path is equal to the subsequence length $L(l_x, l_y)$ as described in Section 4.2.1. Let $\mathcal{P}$ be the set of cells in the warping path. From Equation (4), we have

$$\begin{aligned}v(i_e, j_e) &= \varepsilon L(l_x, l_y) - \sum_{(i,j)\in\mathcal{P}} ||x_i - y_j||\\ &= \varepsilon L(l_x, l_y) - d_{i_s,j_s}(l_x, l_y).\end{aligned}$$

As a result, we obtain the following equation for transforming the score into the DTW distance as regards the subsequence pair,

$$\begin{aligned}d_{i_s,j_s}(l_x, l_y) &= \varepsilon L(l_x, l_y) - v(i_e, j_e)\\ &\qquad s.t.\ v(i_e, j_e) > 0.\end{aligned} \tag{14}$$

On the other hand, from Equations (3) and (14), we have

$$\begin{aligned}d_{i_s,j_s}(l_x, l_y) &\le \varepsilon(L(l_x, l_y) - l_{min})\\ \varepsilon L(l_x, l_y) - v(i_e, j_e) &\le \varepsilon(L(l_x, l_y) - l_{min})\\ v(i_e, j_e) &\ge \varepsilon l_{min}.\end{aligned} \tag{15}$$

From the second condition of Problem 1, it is clear that the optimal warping path from cells (i_s, j_s) through (i_e, j_e) in the time warping matrix starting from (i_s, j_s) gives the minimum distance. From Equations (5) and (14), we also choose the same warping path from (i_s, j_s) in the score and position matrices. Thus, we obtain the conditions of Lemma 1, which are equivalent to those of Problem 1. □

B Proof of Lemma 2

To demonstrate the correctness of CrossMatch, the following three properties should be satisfied.

1. Any reported subsequence pairs must satisfy the property of cross-similarity (i.e., Definition 1).
2. Each reported subsequence pair must be the optimal pair among the set of overlapping subsequence pairs.
3. If a subsequence pair that satisfies the property of cross-similarity is not reported, another overlapping subsequence pair is reported where $D(X[i_s : i_e], Y[j_s : j_e]) - \varepsilon(L(l_x, l_y) - l_{min})$ is the minimum value.

Property 1:
CrossMatch reports a pair where $V(X[i_s : i_e], Y[j_s : j_e]) - \varepsilon l_{min}$ is the maximum value among the overlapping subsequence pairs satisfying $V(X[i_s : i_e], Y[j_s : j_e]) \ge \varepsilon l_{min}$. From Lemma 1, the pair obviously satisfies the property of cross-similarity. □

Property 2:
From Lemma 1, the subsequence pair that minimizes $D(X[i_s : i_e], Y[j_s : j_e]) - \varepsilon(L(l_x, l_y) - l_{min})$ is equivalent to the pair that maximizes $V(X[i_s : i_e], Y[j_s : j_e]) - \varepsilon l_{min}$. We assume that two overlapping subsequence pairs that have different starting positions (i_s, j_s) or (i'_s, j'_s) share cell (i, j). $D(X[i_s : i], Y[j_s : j])$ is the minimum distance in the alignment from (i_s, j_s) to (i, j) of the time warping matrix starting at (i_s, j_s). Similarly, $D(X[i'_s : i], Y[j'_s : j])$ is also the minimum distance in the time warping matrix starting at (i'_s, j'_s). Two pairs share a common warping path in the subsequent alignment from (i, j) to (i_e, j_e) because DTW computes the cumulative minimum distance. Thus, the subsequence pair that minimizes $D(X[i_s : i], Y[j_s : j]) - \varepsilon(L(i - i_s + 1, j - j_s + 1) - l_{min})$ is equivalent to the pair that maximizes $V(X[i_s : i], Y[j_s : j]) - \varepsilon l_{min}$. CrossMatch selects the pair with the maximum score in each cell. Therefore, the matrices that CrossMatch prunes, i.e., time warping matrices that are absolutely not reflected in the score and the position matrices and that are pruned during the computation process, do not include the optimal pair. As a result, CrossMatch constantly reports the optimal pair from the overlapping pairs. □

Property 3:
From property 2, the overlapping subsequence pairs share the same starting position through the operation of CrossMatch. When the subsequence pair satisfying Equation (15) is detected, CrossMatch checks the pair with the same starting position in the candidate array. If the score of the detected pair is greater than that of the pair in the candidate array, CrossMatch updates the candidate pair by using the pair with the maximum score. This process is performed for every pair with a different starting position. Thus, if a subsequence pair that satisfies the property of cross-similarity is not reported, there is another better candidate pair. □

C Proof of Lemma 3

The naive solution has to maintain $O(nw + mw)$ time warping matrices. It updates the $O(w)$ values between x_i and the corresponding elements of Y (i.e., the elements from y_{i-w} to y_i) in $O(nw)$ matrices if we receive x_i at time-tick i. Similarly, it updates the $O(w)$ values in $O(mw)$ matrices if we receive y_j at time-tick j. Therefore, it requires $O(nw^2 + mw^2)$ time per time-tick. Since the naive solution maintains two arrays of w numbers for each matrix, it requires, in total, $O(nw^2 + mw^2)$ space. □

D Proof of Lemma 4

CrossMatch maintains two matrices (i.e., score and position matrices). It updates the $O(w)$ values if we receive x_i or y_j. Each matrix maintains two arrays of w numbers. Thus, it requires $O(w)$ time (per update) and space. □

E Proof of Lemma 5

The sampled sequences compress the original sequences to the size of $1/T$. CrossMatch updates the $O(w/T)$ values, which requires $O(w/T)$ time (per update) and space. □

F Proof of Lemma 6

Given three sequences X, Y, and Z whose lengths are n_1, n_2, and n_3, the naive solution has to maintain $O(n_1 w + n_2 w + n_3 w)$ time warping matrices and updates the $O(w^2)$ values for each matrix. Therefore, it requires $O(n_1 w^3 + n_2 w^3 + n_3 w^3)$ time per update. Since the naive solution maintains two planes of w^2 numbers for each matrix, it requires $O(n_1 w^3 + n_2 w^3 + n_3 w^3)$ space. □

G Proof of Lemma 7

We assume that the warping path from cells (i_s, j_s, k_s) through (i_e, j_e, k_e) includes cell $(i_e - 1, j_e - 1, k_e - 1)$ in each time warping and score matrix. From Equation (12) and $b_d = L(l_x, l_y) - L(l_x - 1, l_y - 1, l_z - 1)$ in Equation (13), we have

$$\begin{aligned}
&||x_{i_e} - y_{j_e}|| + ||y_{j_e} - z_{k_e}|| + ||z_{k_e} - x_{i_e}|| \\
&\quad = d_{i_s, j_s, k_s}(l_x, l_y, l_z) - d_{i_s, j_s, k_x}(l_x - 1, l_y - 1, l_z - 1) \\
&\quad = \varepsilon L(l_x, l_y, l_z) - v(i_e, j_e, k_e) \\
&\qquad - \varepsilon L(l_x - 1, l_y - 1, l_z - 1) + v(i_e - 1, j_e - 1, k_e - 1).
\end{aligned}$$

In the other six neighboring cells, similar equations hold regarding the distance value between three elements. From

$$\begin{aligned}
&||x_{i_s} - y_{j_s}|| + ||y_{j_s} - z_{k_s}|| + ||z_{k_s} - x_{i_s}|| \\
&\qquad = d_{i_s, j_s, k_s}(1, 1, 1) = \varepsilon - v(i_s, j_s, k_s),
\end{aligned}$$

the time warping and the score matrices, which have the starting position (i_s, j_s, k_s), share the same warping path. The sum of the weights on the warping path is equal to the subsequence length $L(l_x, l_y, l_z)$. From Equation (12), we have

$$\begin{aligned}
&v\ (i_e, j_e, k_e) \\
&= \varepsilon L(l_x, l_y, l_z) - \sum_{(i,j,k) \in \mathcal{P}} ||x_i - y_j|| + ||y_j - z_k|| + ||z_k - x_i|| \\
&= \varepsilon L(l_x, l_y, l_z) - d_{i_s, j_s, k_s}(l_x, l_y, l_z).
\end{aligned}$$

Note that $\mathcal{P}$ represents the set of cells in the warping path. Therefore, we obtain the following equation,

$$d_{i_s, j_s, k_s}(l_x, l_y, l_z) = \varepsilon L(l_x, l_y, l_z) - v(i_e, j_e, k_e). \quad (16)$$

Moreover, from Equations (11) and (16), we have

$$v(i_e, j_e, k_e) \geq \varepsilon l_{min}. \quad (17)$$

From Equation (16), the subsequences that have the maximum score are equal to the subsequences that have the minimum DTW distance. We choose the same warping path from cells (i_s, j_s, k_s) through (i_e, j_e, k_e) in the time warping and the score matrices. Thus, we obtain the condition of Lemma 7. □

H Proof of Lemma 8

CrossMatch maintains $2w^2$ arrays (i.e., previous and current planes, which have $w * w$ arrays per plane) for sequences X, Y, and Z in the score and position matrices. It updates $O(w^2)$ numbers to identify the optimal subsequences if we receive x_i at time-tick i, y_j at time-tick j, or z_k at time-tick k. Therefore, it requires $O(w^2)$ time (per update) and $O(w^2)$ space. □